Hands-On Artificial Intelligence and Cybersecurity Education in High Schools

By

Hamid Zare

Thesis

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

MASTER OF SCIENCE

in

Computer Science

December, 14, 2019

Nashville, Tennessee

Approved:

Ákos Lédeczi, Ph.D

Xenofon Koutsoukos, Ph.D

DEDICATION

I dedicate this work to my parents.

ACKNOWLEDGMENTS

This work would not have been possible without the financial support of the National Science Foundation, and Vanderbilt University. I am especially indebted to my adviser Dr. Ákos Lédeczi, who has been supportive of my goals and worked diligently to provide me with personal and academic guidance to pursue those goals.

I am thankful to all those whom I had the pleasure to work and collaborate with during this work and related projects. I would also like to thank Dr. Xenofon Koutsoukos for being on my thesis committee and providing me with professional guidance as well as Dr. Gautam Biswas, Dr. Aniruddha Gokhale, and Dr. Douglas Schmidt for their support.

In addition, I would like to thank all my other colleagues at Vanderbilt University who assisted me during my studies. In particular, I would like to thank Dr. Brian Broll, Péter Völgyesi, Dr. Miklós Maróti, Nicole Hutchins, Gordon Stein, Bernard Yett, Timothy Darrah, Dr. Amin Ghafouri, Mary Metelko, Dimitrios Boursinos, Dr. Zhenkai Zhang, and others at the Institute for Software Integrated Systems.

TABLE OF CONTENTS

Page

DEDICATION . . . . **i**

ACKNOWLEDGMENTS . . . . **ii**

LIST OF TABLES . . . . **v**

LIST OF FIGURES . . . . **vi**

LIST OF ACRONYMS . . . . **viii**

I Introduction **1**

I.1 Artificial Intelligence 3
I.2 CS Education 4
I.2.1 Block Based Visual Programming 5
I.2.2 NetsBlox 5
I.3 RoboScape 6
I.4 Teaching Cybersecurity 9
I.5 What is Next 9

II RoboScape: Provisioning & Access Control **10**

II.1 Provisioning 10
II.1.1 Design and Implementation 13
II.1.2 Authority: The App 14
II.2 Access Control 15
II.2.1 Robot Ownership & User Management 17
II.2.2 Robot Ownership Verification 17
II.2.3 Model Design 18
II.2.3.1 Enforcing Control 19
II.2.4 Access Management - UI 20

III Visual RL **22**

III.1 Background 23
III.1.1 Related Work 23
III.2 Design and Implementation 24
III.2.1 RL Framework and the Blocks 29
III.3 Experiments & Results 30
III.4 Future Work 35
III.5 Conclusion 35

IV NetsBlox Player **36**

IV.1 Methodologies and Implementations 37
IV.2 Authentication & Projects 38
IV.3 Typing Support 39
IV.4 Key Listeners 40
IV.5 Room & Networking 40
IV.6 Multiple Platforms & Servers 41

V Discussion & Conclusion **42**

BIBLIOGRAPHY **43**

Appendix A Q Learner Framework **45**

LIST OF TABLES

Table Page

II.1 Robot schema. 18

II.2 Robot-User association schema. 18

LIST OF FIGURES

Figure		Page
I.1	Growth projection of connected devices. [Schwab, 2015]	2
I.2	Hello world in a block based programming language.	5
I.3	Message passing blocks in NetsBlox.	6
I.4	”getMap” RPC from Google Maps service.	6
I.5	The weather application in a block based programming language (NetsBlox).	6
I.6	RoboScape architecture: clients communicate with robots through NetsBlox server.	7
I.7	A simple RoboScape project for driving the robot with a keyboard.	8
II.1	Original RoboScape robots Parallax Activity Bot 360.	10
II.2	RoboScape provisioning diagram.	14
II.3	Authentication view.	16
II.4	Main view.	16
II.5	Configuration view.	16
II.6	Different RoboScape provisioner app views	16
II.7	RoboScape ownership verification process.	18
II.8	Individually adding users for controlled access.	20
II.9	Controlling access on a per robot basis.	20
II.10	Adding and viewing user access.	20
II.11	Robots under the user’s control.	20
III.1	Q-learning procedural algorithm.	25
III.2	Q-table. Source [LearnDataSci, 2019].	26
III.3	RL framework interactions within NetsBlox	28
III.4	Core reinforcement learning blocks (Q-learning)	28
III.5	Utility blocks for saving the agent and working with the Q-table.	30

III.6 A basic learning scenario using Visual RL 30

III.7 Grid world game interface 31

III.8 Building blocks of grid world project. 32

III.9 Grid world reward function. 33

III.10 Number of steps taken by the grid world agent per episode over the course of learning. . . 33

III.11 Pet trainer interface 34

IV.1 Smartphone usage growth versus PC and laptop. 36

IV.2 Tablet or small desktop screen 37

IV.3 Project view. 39

IV.4 Key listener blocks 40

IV.5 Room management in NetsBlox Player. 41

ACRONYMS

AAA Authentication, Authorization, and Accounting. 17

AI Artificial Intelligence. 3, 9, 22, 24

AP Access Point. 11, 14

API Application Program Interface. 24, 38

CSS Cascading Style Sheets. 15, 37

DDOS Distributed Denial of Service. 19

DOS Denial of Service. 19

GPS Global Positioning System. 38

HTML Hyper Text Transfer Protocol. 37

HTTP Hyper Text Transfer Protocol. 14

IoT Internet of Things. 11

MDP Markov Decision Processes. 35

NLP Natural Language Processing. 24

PSK Pre-shared Key. 12, 15

SDK Software Development Kit. 13

SSID Service Set Identifier. 12, 15

WPS Wi-Fi Protected Setup. 12

CHAPTER I

Introduction

Cybercrime is on the rise, and there is a serious shortage of people to fight it. Cybercrime is one of the greatest hazards to businesses around the world. As more and more parts of our lives become intertwined with computer systems, the potential negative impacts of cybercrime rise exponentially. Damages from cybercrime are anticipated to exceed $6 trillion by 2021 up from $3 trillion in 2015 [Morgan, 2019]. This represents the greatest transfer of financial wealth in history and could be more profitable than all other forms of illegal activities. The damage projections caused by cybercrime are based on historic figures which include recent year-over-year growth, with a dramatic increase in the state-backed and organized hacking activities, and a cyberattack surface an order of magnitude greater in 2021 than it is today.

The lack of professionals in the field to counter cybercriminals exacerbates this situation. Many people assume that cybersecurity is only needed to protect top-secret army plans or government secrets, however, the fact is that any information that you would not want to tell a complete stranger is in need of protection. The scarcity of qualified cybersecurity personnel is getting worse every year. While this is bad news for the cyber wellbeing of people and companies, it does make cybersecurity a relatively rewarding career path with a lot of job security, for those who are interested in pursuing it. The shortage of certified cybersecurity personnel is visibly obvious with the number of unfilled cybersecurity roles on the rise. The demand for skilled cybersecurity experts will continue to exceed reachable resources with the prediction of 3.5 million unfilled positions globally by 2021 [Morgan, 2019].

America and the world need well-trained educated people working in cybersecurity roles. These specialists are crucial in both private enterprise and the government for the security of men and women and the nation. Institutions including government agencies are realizing this need and are dedicating resources to strengthen the nation's cybersecurity by ensuring there are well-trained cybersecurity experts today as well as a sturdy pipeline of future cybersecurity professionals of tomorrow.

It is not hard to see the importance of cybersecurity; one only needs to open the newspaper to read about new security incidents. The Yahoo hack was recalculated to reveal it had affected all 3 billion user accounts. The Equifax breach in 2017, with 145.5 million clients affected, exceeds the biggest publicly disclosed hacks ever reported. These incidents alongside others, such as the WannaCry and NotPetya, ran on a larger scale and caused greater complications than preceding attacks. If there is one thing we know about cyberattacks it is that they will only get worse, not better, because of the rapidly expanding attack surface into our digital lives.

The World Wide Web was invented less than thirty years ago and there were more than 1.2 billion websites serving about four billion Internet customers in 2017 (more than half the population of earth), up from 2 billion in 2015. Cybersecurity Ventures [Morgan, 2019] predicts that there will be 6 billion Internet customers through 2022 (three-quarter of the projected world population of eight billion) and upwards of 7.5 billion Internet customers by 2030 (90% of the world population).

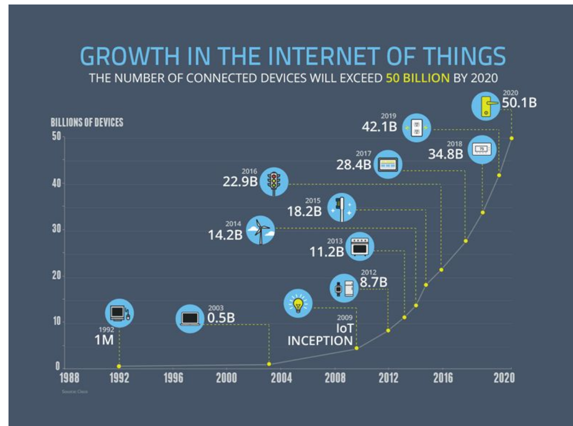

Figure I.1: Growth projection of connected devices. [Schwab, 2015]

World Economic Forum predicts (Figure I.1) there will be 50 billion connected devices by 2020 while Intel predicts that number to be 200 billion [Intel, 2019]. ABI has forecast that more than 203 million Software Over The Air (SOTA) enabled vehicles will ship by 2022 [ABI, 2016]. Hundreds of thousands of humans can be hacked now via their wirelessly linked and digitally monitored implantable medical units which encompass pacemakers, neurostimulators, insulin pumps, ear tubes, and more [Morgan, 2019]. Gartner forecasts that more than half a billion wearable units will be sold worldwide in 2021, up from roughly 140 million in 2017. Wearables including smartwatches, head-mounted displays, body-worn cameras, Bluetooth headsets, and health monitors [Stamford, 2018]. Add all that with the 111 billion new lines of code produced every year [Morgan, 2019] which introduces a massive number of vulnerabilities that can be exploited. Our entire society is connecting up to the Internet at a rate that is out-pacing our potential to fix all the vulnerabilities that come with it.

Cybercrime causes damage and destruction of data, stolen money, misplaced productivity, theft of mental property, theft of private and economic data, embezzlement, fraud, and not to forget harming reputation. Cyberattacks are the quickest developing crime in the US and they are increasing in size, sophistication and cost [CNBC, 2017]. In the face of all these threats, companies and organizations that are searching for long-term solutions to cyber incidents are finding funding cybersecurity education and employing an expert team of professionals a more competitively priced option. In addition, everyone needs to have some cybersecurity

knowledge since even with the best cybersecurity experts and security measures in place, the weakest link in any organization, when it comes to security, is frequently the employees.

As more emphasis is placed on strengthening cybersecurity, expectations turn toward the education system to train students to become capable cybersecurity experts to fulfill the ever-increasing demand for the fight against cybercrime and to protect our networks and infrastructures. Focusing on the country's youth, government agencies and organizations, such as Department of Homeland Security (DHS), National Integrated Cyber Education Research Center (NICERC), National Centers of Academic Excellence (CAE), National Science Foundation (NSF), and National Security Agency's GenCyber, have partnered with not-for-profits centers, schools, universities, and faculty boards throughout the United States to help include cybersecurity curriculum into the nation's classrooms. There are also Scholarship for Service (SFS) programs providing financial aid for bachelor's and master's degrees focusing on cybersecurity in return for service in government upon graduation.

I.1 Artificial Intelligence

AI is changing everything, but who is going to change AI?

AI has become dramatically more prevalent in the last few years. The financial impact of AI worldwide is anticipated to reach $15 trillion by 2030 [PWC, 2019] and we're already seeing AI integrated into areas like medical diagnosis, self-driving cars, and automated customer service. Young people of this generation may be digital citizens but most of them are digitally naive when it comes to how technologies around them work and few of them respect that AI is reshaping the landscape and social norms. Most people are unaware that AI is a key field behind private assistants (Alexa, Siri, Google), smart vehicles, predictive analytics (Amazon and Netflix recommendations), and medical diagnostics just to name a few.

It is predicted that AI will create 58 million new jobs by 2022 [Schwab, 2018]. It is important for the new generation to be conscious consumers of AI and for some to develop expertise in the area to be able to extend it further. Shouldn't everyone, especially young people, understand one of the biggest forces impacting their lives and livelihoods? They want to be part of the dialog about the technologies that are shaping their future. Understanding how AI works, the data used to train systems, and the benefits and pitfalls of AI are fundamental knowledge to be an active participant in these discussions. However, there is a lack of diversity in AI. Homogeneous teams of professionals are constructing AI solutions for everyone and women and minorities are being left out. Lack of variety in the field can result in biased AI products that in the best case only serve a specific group or at worst, adversely affect minorities. It is not hard to imagine algorithmic and social biases creeping into smart systems and products being developed.

Two important barriers to getting into the field of Artificial Intelligence (AI) are access and interest. The

access barrier comes from a lack of awareness and exposure to effective educational material and opportunities. Even with access to such programs, oftentimes interest plays a big role in the final decision making for students and parents to make the decision to invest in AI and computer science courses. A lack of exposure to technical concepts early on, missing mentors and lack of peer to peer help are big contributing factors to this issue [Margolis et al., 2000]. Students question their belonging in new environments and learning new topics is no different, especially when there is a stigma or stereotypes around the subject matter. Research suggests self-efficacy has the biggest influence on STEM entrance and persistence within the field [Wang, 2013].

It is important to act now to make sure that all of us have the possibility to drive the introduction of AI as a force for good. The initial step would be to help students recognize that AI is around them and help them understand the effect it is having on their lives. When they can identify how it is changing their everyday lives, they will see why it is significant and that is a great inspiration for finding out more about it.

A comprehensive AI education plan, as AI for K-12 white paper [Gardner-McCune et al., 2019] sees it, outlines five big ideas: What should these educational plans resemble? They identify five key ideas. On the technical side, we should guide students to discover that computers see the world using sensors to collect data which then they can use to make reasoning models and become "intelligent." Using complex algorithms, artificial intelligence systems derive new statistics and knowledge from existing data points that are often furnished or crafted by humans. On the moral side, the program should help students see the challenges in AI as well as how it can affect society in positive and negative ways. It is important to show how AI is contributing to our lives. When do we need these systems and what moral criteria should they meet? There are great study plans and courses built around these ideas. In this work, we focus on improving the tools available to teach the core concepts of AI to young learners in a way that is accessible and welcoming to everyone. By choosing NetsBlox as the base visual programming environment, we bring all the interesting aspects of networked, Internet-enabled, block-based programming platforms and mix it with AI concepts. We believe that this would not only help students understand how AI is changing their world, but it may also spark their interest so that one day, they will be the ones changing AI.

I.2 CS Education

Computational Thinking (CT) is a skill that involves solving problems, designing systems and understanding human behaviors through fundamental computer science concepts [Wing, 2006]. CT is receiving increasing attention as institutions realize its importance and the fundamental role it plays in today's society [Wing, 2006]. Different curricula are incorporating programming and CT concepts into their programs [Resnick et al., 2009; Kafai and Burke, 2013; Grover and Pea, 2013]. Many of these programs take advantage of visual programming environments, more specifically block-based languages.

Block-based languages inherently eliminate the syntactic issues that novice programmers face when learning new computing concepts. In these environments, the visual characteristics of the blocks, such as their shape or color give clues about the usage and behavior of the block, encouraging exploration and reducing errors. Blockly [Fraser, 2015], code.org, Scratch [Resnick et al., 2009], Snap! [Harvey et al., 2014], NetsBlox [Broll et al., 2018], and App Inventor [Wolber et al., 2011] are all examples of such environments. Throughout this thesis, we explain four of the main contributions to CT education using blocks-based programming languages in general and NetsBlox in particular.

I.2.1 Block Based Visual Programming

Incorporating visual programming into the CS curriculum is an effective way of teaching programming and computer science concepts while hiding unnecessary syntactical complexities. In these block-based environments, users provide program instructions in the form of blocks. Figure I.2 shows the classic hello world in Snap!. As a side benefit, these environments provide a set of tools to create visual elements on the screen which helps with creating animations or building games. Video games are a popular choice of project for students who are getting acquainted with computer science for the first time, which lends itself very nicely for teaching AI through reinforcement learning as shown in Chapter III . Before we discuss the contributions of this work, we first take a brief look at NetsBlox and RoboScape as these form the foundation for this thesis.

Figure I.2: Hello world in a block based programming language.

I.2.2 NetsBlox

Computer networking and distributed computing concepts have become an important part of computer science and modern life in general. These technologies are everywhere from social networks, through online multi-player games, to self-driving cars. The lack of educational tools to support effective teaching of key underlying concepts of distributed computing and computer networking to high school students lead to the creation of NetsBlox. NetsBlox extends Snap! with a few selected abstractions, including Remote Procedure Calls (RPC) and message passing, to allow for the creation of distributed applications. The message passing primitives are shown in Figure I.3.

RPCs provide access to free online data sources and services such as Google Maps, weather, movies or plotting. Figure I.5 shows the map and weather services come together to create a live weather application. RPCs in NetsBlox are much more than just calls to web APIs; they are simpler to use and they keep state

Figure I.3: Message passing blocks in NetsBlox.

Figure I.4: "getMap" RPC from Google Maps service.

information. Related RPCs are grouped into services and are logically separated as shown in Figure I.4. Some services even send messages to the user's program.

Message passing blocks are similar to events in Snap! with the addition that they carry data and can be sent to remote targets (other NetsBlox programs across the network/internet). These two abstractions open the door to teaching some of the fundamental concepts of distributed computing such as communication protocols, shared state, synchronization, addressing, latency, message loss, etc.

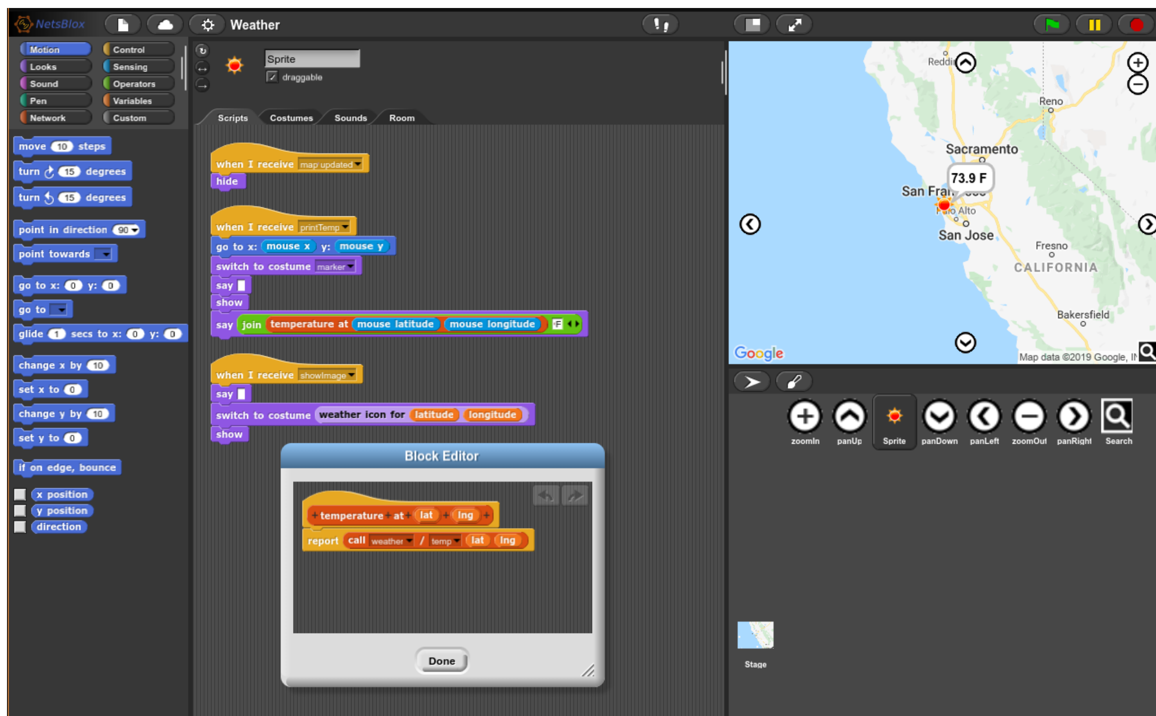

Figure I.5: The weather application in a block based programming language (NetsBlox).

I.3 RoboScape

The use of robots in just about any topic makes it more interesting to kids and that is just one of the reasons that utilizing robots, makes for an excellent platform to teach computer science and engineering in K12. Traditionally, educational robots are programmed using text-based languages like C or Python, then the

program code is compiled and downloaded to the robot. These steps can add unnecessary complexity, and make the development cycle cumbersome and slow down the debugging process. Would it not be great if programming robots was as easy as programming sprites in Scratch or Snap!?

RoboScape is a new way to program wireless networked educational robots. It is implemented as a NetsBlox service and benefits from all the features that come with it. RoboScape uses Remote Procedure Calls (RPC) and message passing, to program and interact with robots. The program runs in the browser remotely controlling the robots with RPCs. The robots can respond to these RPC by taking actions and sending sensor values back as NetsBlox messages to the user. This approach makes programming robots as easy as programming sprites. Leveraging the integration with NetsBlox, RoboScape provides the means to control robots and handle sensory feedback by utilizing existing abstractions in NetsBlox. The student's code executes entirely in the browser, hence, there is no need to download anything allowing the student to easily verify their logic by thinking through the problem as if they were programming the sprites to move on the stage. In addition, the users can take advantage of the debugging features that NetsBlox and Snap! provide such as stepping through the code, lively execution, variable inspection, and more.

Since all the robots are programmed remotely, programming for multiple robots or targeting another robot from your program is easy. If desired, access to multiple robots can be granted to the students allowing them to control as many robots as they want from the same program.

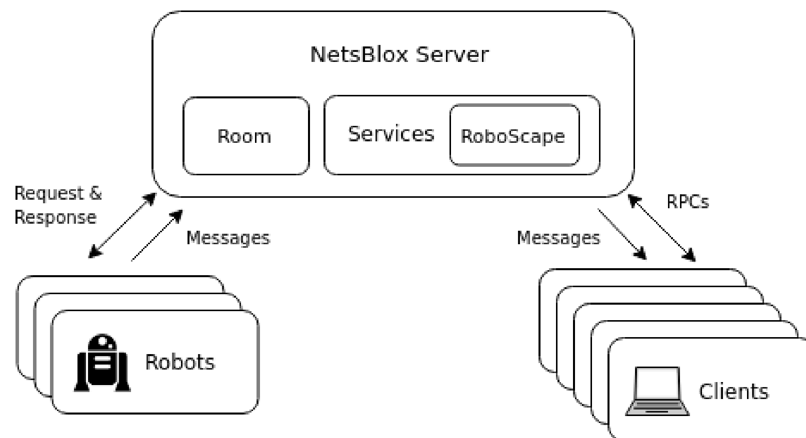

Figure I.6: RoboScape architecture: clients communicate with robots through NetsBlox server.

The RoboScape service handles the communication between the users and the robots. This service includes calls to query sensors as well as to control the robot behavior. Figure I.6 paints the RoboScape architecture. Each call takes a robot ID as its first argument making targeting different or multiple robots as easy as changing a single id value. A simple RoboScape program for driving the robots can look like Figure I.7. Our first version supports the Parallax ActivityBot 360 [Parallax, 2019] robots. Some of the RPCs include:

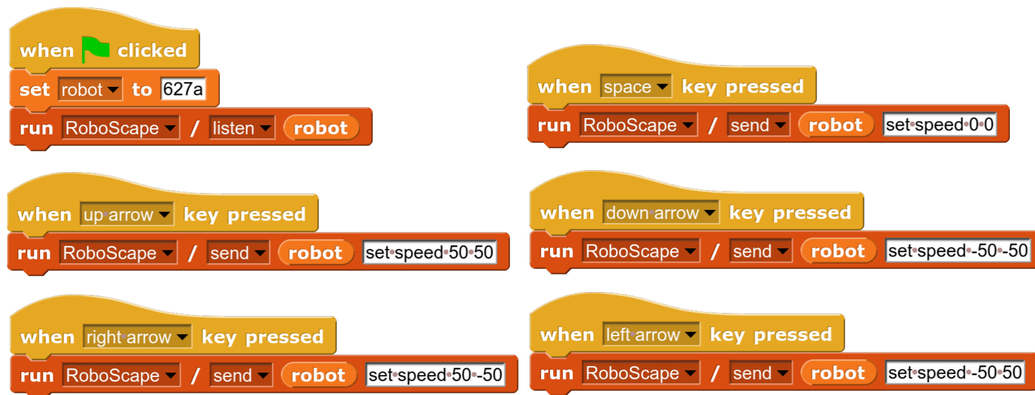

Figure I.7: A simple RoboScape project for driving the robot with a keyboard.

- is alive: check whether a robot with a given ID is alive and responsive.
- beep: make the robot's buzzer module beep.
- set speed: set individual wheel rotation speed.
- get range: calls the ultrasonic range sensor module to probe and return the range.
- get ticks: returns the number of ticks for each wheel counted by the optical encoder.
- listen: subscribe to messages from the robot.

In addition to replying to user requests, the robots can also push messages to the server, some of which end up going to the clients. If the user wants to receive these messages they will first subscribe to receiving updates from the robot through the "listen" RPC and then create the messages handlers for target messages. These include:

- Identification message: sent every second and is used to ID the robot and keep the connection alive.
- Range: whenever the "get range" RPC is called, a message with the sensor value is sent (in addition to being sent as the return value of the RPC call).
- Button pressed: updates about individual button presses which can then be used to determine different button press patterns, e.g., short or long-press.
- Whiskers: informs the client every time the robot's whiskers are pushed or released and can be used to detect obstacles.

I.4 Teaching Cybersecurity

RoboScape also brings about a great opportunity for teaching cybersecurity. It exposes a security mode where most of its RPCs are replaced by a single one called *send*. This is a necessary change since in the security mode, the robots support different encryption algorithms and students, once ready, can start utilizing those. The two arguments of *send* are the robot id and the textual command to send. This version of RoboScape uses only two message types: *robot command* and *robot message*. They each have two fields: the robot id and the command (or message). Robot commands, sent from a program to the robot, and robot messages, sent from the robot to the user, can be overheard by other users. Since they are sent in clear text, students can eavesdrop on each other and even hijack each others' robots. This is of course simulated: the NetsBlox server sends NetsBlox messages to all the clients that are listening to a given robot.

After going through the first few RoboScape lesson plans and experiencing interference from other classmates, students quickly realize the need for securing their robots. This provides an excellent entry point for talking about cybersecurity, its importance, and why and how they can secure their robots. The very first approach to securing their robot will usually start with encryption. RoboScape supports different encryption algorithms that can be used to secure user-robot communication. When the robots are booted they are not using any encryption. The clients can issue a "set key" command to set up the encrypted communication that makes the robot and their program agree on an encryption algorithm and a shared key. By default, the encryption algorithm is set to a simple substitution cipher called Caesar cipher. Eventually, the students realize that key exchange should not happen in the clear either. A solution to remedy the in-band key exchange issue is to move that process out of band. There is a button on each robot that auto generates a key and displays it by toggling its two LEDs on and off. This randomly generated key will be the initial encryption key transferred through a secure, private medium as opposed to the shared and exposed wireless network. More ambitious students can then progress to breaking the cipher using brute force or other methods. Other modules in a RoboScape lesson plan can include replay attacks and defend against it by utilizing sequence numbers or denial of service attacks and their remediation by rate-limiting.

I.5 What is Next

In the following chapters, we will first look at two contributions to RoboScape starting with how the robots are provisioned and set up in Section II.1, followed by access control for said robots in Section II.2. Next, we look at Visual RL (Section III), a set of reinforcement learning abstractions and blocks for Snap! which aims to improve hands-on Artificial Intelligence (AI) education for young programmers. Last but not least, in Chapter IV we bring NetsBlox projects to smartphones by introducing NetsBlox Player, a native execution system for today's IOS and Android devices.

CHAPTER II

RoboScape: Provisioning & Access Control

II.1 Provisioning

RoboScape started with a dozen robots from Parallax called Activity Bot 360 [Parallax, 2019] shown in Figure II.1. As described in Section I.3, the robots are programmed and controlled remotely through the network thus they cannot operate if they are not connected to the network or cannot reach the NetsBlox server. The robots were designed to connect to a known and always available access point using pre-set credentials. But this approach poses limitations to where the robot can be used. Either an access point with Internet up-link has to be prepared, carried around, and set up wherever the robots are to be used, or the robots' firmware had to be flashed each time to provide the required information for the robot to connect to a new access point. Neither of these is ideal.

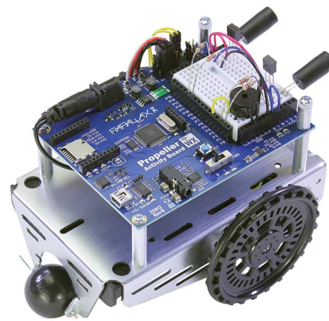

Figure II.1: Original RoboScape robots Parallax Activity Bot 360.

Often there is a need to have the robot connect with a different access point; be it a teacher moving the robots to different classes, a student taking the robot home, a developer testing the robot in a different environment, a researcher taking the robot around for demoing purposes, etc. In addition, in a classroom setting, the teacher is managing anywhere between one to a hundred different robots and re-configuring all to connect to a new access point, either by re-flashing the firmware or using a provisioning technique can be tedious, error-prone, and time-consuming.

To come up with suitable solutions to these problems and be able to evaluate them, first, we need to clearly define the requirements. We laid out the initial set of requirements as the following:

- Ease of use:

  - Target audience is students and teachers
  - No need for flashing the firmware

- Be able to instruct the robots to connect to any compatible Access Point (AP).
- No need for specialized hardware, it should work with commodity devices.
- Free or low cost.
- Relatively fast.
- Able to configure many robots at once or autonomously with granular control.
- No or low user intervention required: fewer errors, and less time spent.
- Easy to expand: Enable the possibility to configure other parameters on the robot for future use.
- Stable and reliable.
- Provide timely feedback and progress reports.
- Persistent configuration: Robot should remember the connection details over reboots.

The problem of setting up networked devices with limited input and output is one faced by many Internet of Things (IoT) vendors. Smart appliances, smart light bulbs, smart plugs, smart speakers, or a networked temperature sensor all deal with this process one way or the other. During this initial provisioning step, someone or something, needs to configure the device to connect to the local network and eventually the Internet, usually through an access point in the room. The robots used for RoboScape are no different. However, a major difference of requirements here compared to typical IoT applications is that we need to provision many robots at once and not just one. Theoretically, provisioning multiple devices could take a few forms that we can breakdown using common networking terminology: unicast, broadcast, and peer to peer.

To explain the different approaches we first define the entity holding the new access point information as the authority and the devices in need of configuration as robots. In a unicast approach, the authority connects to each robot one by one and directly communicates the required information, mainly the Wi-Fi credentials. In a broadcast scenario, after all the robots and the authority are in the same broadcast domain, the authority broadcasts the information. Giving all the participants a shared communication channel for this phase needs some preparation. The robots could know about a pre-defined portable access point where they will start looking for whenever they fail to connect to their primary AP. This AP could be a portable one that is used just for provisioning purposes. Modern smartphones could be a good candidate for always having such an

access point on hand. But unfortunately, neither IOS nor Android provide the necessary controls to realize this solution. IOS completely prevents programmatic control of its hotspot feature and android does not officially expose an API to pick a hotspot name and randomly generates one upon request.

Finally, we have the peer to peer approach. The idea here is for the authority to delegate the configuration responsibility to other participants, namely robots. Initially, the authority configures one of the robots directly and then they start serving each other until some condition is met and the configuration phase is over. More robots can be set to start serving configuration to others to speed up the process. This approach can be summarized in the following steps:

1. Transition to configuration mode: either automatically by failing to successfully connect to the original AP or manually through a push of a button.

2. Scan for and connect to a pre-defined AP with known credentials.

3. If failed to find such an AP, turn into a master node and create the expected AP.

In general, all these ideas share the same two steps: First creating a communication channel and second transmitting the configurations either in a pull or a push mechanism. We considered the following solutions for providing a communication channel between the authority and the robots. One of the initial ideas was to make the robots [Viswanathan et al., 2015] Wi-Fi Protected Setup (WPS) compliant and utilize the WPS button on the access points. This idea, however, was discarded quickly because of two main issues. First was the reduced availability of this feature in modern routers, either by being disabled by default or lack of support. The security flaws discovered surrounding WPS [Viehböck, 2011] has been the main contributor to its reduced popularity. In addition, the need for physical access to push the WPS button, or administrative access for virtual buttons, made it unusable for most classroom settings.

In another approach, we would rely on the authority to always provide a wireless access point with predefined Service Set Identifier (SSID), encryption algorithm, and Pre-shared Key (PSK) during the setup phase of the robots. The robots can be programmed to look for and join this access point during this transition period, to use as the network medium. Finally, dependent on the capabilities of the Wi-Fi module used in the robots, one of them could take on the role of authority by initially starting out as a wireless access point. Some options for setting up this communication channel are using a microcontroller such as ESP or Arduino, a smartphone, or a Linux box. Once there is a communication channel established, following a simple discovery protocol the robots can start talking with the authority, or authorities, to pick up their configuration. How a teacher can provide the credentials of an access point or any other configuration settings, could vary based on how the initial communication channel was set up. Options for a UI could be a web-server on the same network as the access point or a native application on a smartphone.

While researching solutions for this problem we discovered some tools and processes worth mentioning:

- Texas instrument's SmartConfig [Instruments, 2017]: It is a proprietary solution capable of configuring multiple TI devices at the same time.
- ESP Touch [Espressif, 2018] provides a phone application and a Software Development Kit (SDK) for configuring its microcontrollers.
- Pfod simple configuration for Wi-Fi devices. This is an Android-only approach. It uses a QR code attached to the devices to identify them.

After considering the viability of the discussed solutions and available resources, we decided upon an android based solution relying on the soft access point capability of Digi XBee S68 [N/A, 2000] modules used in the robots. In this approach after each robot decides to go to its configuration mode, it will set up an access point. The owner of the robots opens up the RoboScape application on their phone to monitor the list of robots in configuration mode. They will then select the robots they want to configure, type in the parameters and initiate the process. The phone then connects to each robot, one after the other, and configures them. The Wi-Fi modules we used with the Activity Bots support a soft access point mode through which they serve a webpage for provisioning the network credentials through a browser. We use this feature to set up the communication channel. To give users a UI for configuring the robots in batch we developed a smartphone application for the Android platform. We picked Android due to some IOS limitations and wide availability of Android enabled devices:

- Android has the highest market share in smartphones. Thus making it very likely that there is no need for purchasing a new device for provisioning purposes.
- A smartphone app is easier to operate compared to custom devices, thanks to their big touch screen and familiar interfaces.
- Mobile applications are easy to install.
- Android devices are cost-effective compared to other smartphones.
- iPhone support is not an option since IOS does not provide enough control over the phone's Wi-Fi module.

II.1.1 Design and Implementation

To support being provisioned the robot firmware needed two main new functionalities: 1. Defining the configuration mode, including bootstrapping of the access point. 2. A detection mechanism or a manual

override for switching to this mode.

In order to add a trigger mechanism for the configuration mode, we set up a physical push button on the robot. Once the robot detects a specific button press pattern, calculated using its internal clock, it initiates the setup process with audible feedback to notify the user. Currently, this pattern is defined as a five-second long button press. More intelligent triggers can also be programmatically defined which we will get into more details in the future work (Section V).

When configuration mode is initiated the robot instructs the XBee Wi-Fi module to stop scanning and go into soft Access Point (AP) mode and present its network setup web form. In this form, the soft AP only accepts a single client, the one supposed to supply the new access point details. The presented form also allows for capturing custom inputs that can be used to configure arbitrary parameters on the robot. After this form is submitted, the Wi-Fi module goes back into station (client) mode and attempts to connect to the newly provided wireless access point. If the attempt fails, the process restarts and it goes back to the soft AP mode. Figure II.2 depicts this process.

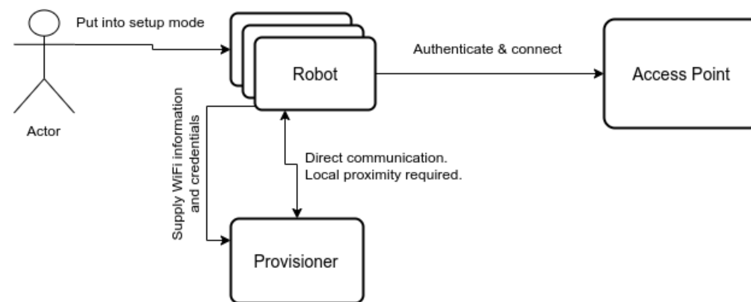

Figure II.2: RoboScape provisioning diagram.

II.1.2 Authority: The App

The other side of our solution is the Android application acting as the authority. This application, at the very least, needs to be able to perform the following steps for a minimum viable product:

1. Find and display available robots to configure which means scanning the available wireless access points in the area.
2. Get the desired configuration from the user
3. Connect to a robot
4. Communicate with the robot
5. Make Hyper Text Transfer Protocol (HTTP) requests to the robot

6. Disconnect from the robot

7. Connect to the next one and repeat

For developing the application we picked Apache Cordova [Cordova, 2009]. Cordova is an open-source hybrid mobile application development framework. It allows the use of web development technologies to create native mobile applications or repackaging the front-end of a website into a native mobile application. It also provides the benefit of writing the application once for the Cordova and running it on different platforms such as IOS, Android, and Windows Phone. However, its support for hardware features varies from platform to platform. In order to control system-level settings and hardware features in Cordova, we took advantage of some already available plugins written by its community. The abstractions provided by Cordova and the plugin systems let us use a single code base to create applications compatible with different platforms including different versions of Android. Some of these system-level functionalities we needed for this project are:

- Scanning local Wi-Fi for information such as signal strength, encryption, and frequency.
- Connecting to known and unknown access points and probing nodes on the network.
- Disconnecting from an access point.

Once Cordova and the underlying logic were set up, creating a UI was as straightforward as creating a mobile-first web site. For a modern mobile design, we chose Framework7 as our frontend Cascading Style Sheets (CSS) framework alongside Vue.js.

In the first view of the application users are presented with a list of suitable and available wireless access points in range of the phone alongside a list of robots ready to be configured. The different views of the application are visible in Figure II.6. They can then proceed to the configuration page where they see different fields for configuring the robots including: SSID, PSK, encryption type, and a live list available robots. After the inputs are filled and robot selections are made, the user can give the go to instruct the application to initiate the process. This view keeps scrolling as the application pushes up progress and status updates. The robots, one after the other, connect to the provided access point. Their successful connection can then be confirmed through NetsBlox.

II.2 Access Control

Robots in RoboScape are, by design, exposed to all the users on the platform. This means that anyone able to access and open the NetsBlox editor has access to all the robots connected to the platform and server. This

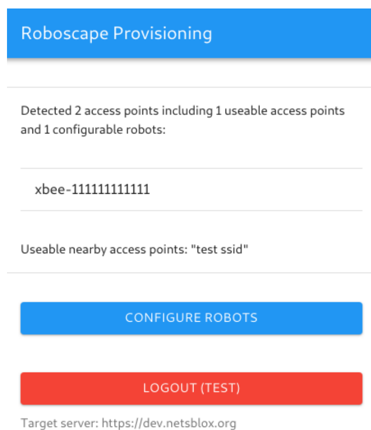

Figure II.3: Authentication view.

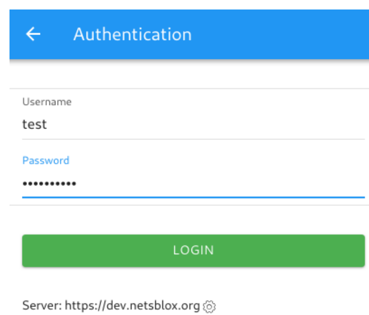

Figure II.4: Main view.

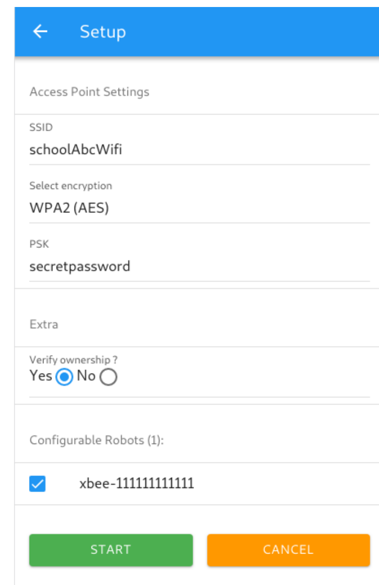

Figure II.5: Configuration view.

Figure II.6: Different RoboScape provisioner app views

unlimited access makes it very easy for educators and students to get started with new robots and/or a new classroom. In addition, the open access motivates cybersecurity ideas discussed previously I.3.

But this openness also means that visitors outside a classroom could access and issue commands to a classroom's robots which could interfere with the class agenda. For NetsBlox setups where the server is local to the users in a classroom or an institute, this is less of a problem since the server is isolated to a select number of people, for example, a classroom but usually, that is not the case. On the other hand, if the robots are connected to the main NetsBlox server or any publicly available server, they are wide open to anyone who works with the server.

To address this issue and give more control to instructors we need to regulate access to the robots while maintaining the ease of use of RoboScape and allowing for the same openness that a classroom teaching cybersecurity would need. Setting up and managing users and robots need to be intuitive and easy to use by teachers and instructors in a classroom setting. Access control needs to be transparent to the users and ideally not require them to go through additional steps to benefit from it. It should be setup up once and remain in effect for as long as the classroom and the robots are around. To make getting started with RoboScape simpler, the access control should be an opt-in feature enabling easy setup for testing and demoing purposes. Along with the access control regulation, there is a need for an easy to use interface which allows the instructors to view and manage the robots and their accessibility by the students.

II.2.1 Robot Ownership & User Management

For a robust access control solution we need to think about Authentication, Authorization, and Accounting (AAA). We initially considered different ways of providing this much-required control. One approach was to use a shared key set for a group of robots. The instructors would set up and share the keys with the students for each classroom. The users then provide the keys with each call they make to the server. From here forward we could take two routes: have the server keep track of active keys for a group of robots, or design it to pass along the keys to the robots on each call and delegate authentication to them. The authentication mechanism, in this case, is simple and there is not much need for accounting. This design would also relieve the server of having to keep much state as the keys are provided with each call resulting in a more fault-tolerant service on the server-side. Besides, this change decouples RoboScape access control from the existing NetsBlox authentication and user management infrastructure. On the other end of the spectrum, we could start by designing different account levels, groups of users (classrooms), and groups of robots and then defining associations between those new entities.

We decided on a hybrid approach between the two to keep the solution simple yet flexible enough to easily meet all the requirements. NetsBlox already provides a form of authentication for users through its user accounting system. We use the same authentication mechanism to identify users for RoboScape use. Before we can do any accounting, we need to define the entities in the system to keep track of who can control and/or access which assets. Once set up for access control, the robots start off with the public access level. The robots can be configured into different security modes (access levels):

- Public mode: There is no access control applied to the robot.
- Protected mode: Only authorized users can see and communicate with the robot.

II.2.2 Robot Ownership Verification

Like many other computing platforms, we see physical access to the robot as having ownership of the device. Considering the application of RoboScape and its audience, this assumption means that we can physical proximity in the ownership verification process. In Section II.1 we talked about robot provisioning and the RoboScape Android app. That process required pushing a physical button on the robot. Since getting the robots into this mode and communicating with it requires physical proximity and access to the robots, it proposed a good opportunity for ownership verification. We modified the RoboScape provisioning application to require a NetsBlox user account. Once the user is logged and the robots are in configuration mode, the user can choose to also perform the ownership verification step and after solving a challenge with the server take ownership of selected robots. Figure II.7 shows this process.

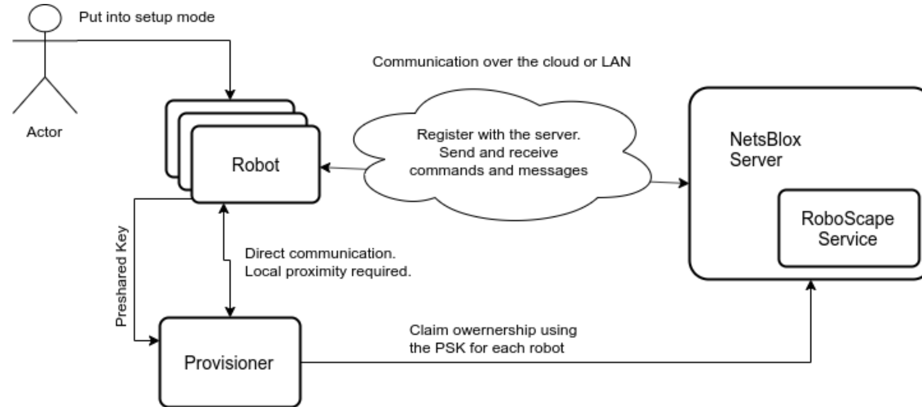

Figure II.7: RoboScape ownership verification process.

II.2.3 Model Design

The access control system is modeled in the following way: Robots have their own presence on the server with different access modes. They can be set to the protected or public mode, which in turn determines their accessibility by registered users or guests. Tables II.1 and II.2 describe database records for each robot and its associations with NetsBlox users.

attribute	type	notes
robotId	String	unique robot identifier
isPublic	bool	access mode
owner	String	owner's username (unique)
ownedAt	Date	time when ownership verification happened
users	RobotUserAssociation	defines user associations with this robot

Table II.1: Robot schema.

attribute	type	notes
username	String	username (unique)
hasAccess	bool	indicates access
updatedAt	Date	when access was modified

Table II.2: Robot-User association schema.

Users can register robots as their own and get permission to regulate access to them and also modify their security mode. Any user can own multiple robots but each robot can only have one owner at a time which lends itself nicely to an exclusive one-to-many relationship. Defining the user-robot relationship as such gives us enough flexibility to meet all the requirements while keeping the design simple. The decision to store the user-robot association on the robot was made based on

- The number of existing users in the database

- The number of existing robots in the database
- The potential number of users in the database
- The potential number of robots in the database
- The existing database family. SQL vs NoSQL
- How frequently users-robot association is going to be accessed dictating read queries
- How frequently users-robot association is going to be modified dictating write queries

The most frequent queries having to deal with access control are by far read queries that are triggered when a command from the client is issued. This is amplified in cybersecurity modules where the users are instructed to practice Denial of Service (DOS) and Distributed Denial of Service (DDOS) on robots. Therefore the database and the corresponding data models are designed to provide the best performance for this scenario to avoid the bottleneck. In comparison, modifying user access to the robots in a typical classroom scenario is very infrequent. By keeping the association information close to the robot we put less strain on the database. In the future, if there is a need for higher performance, we can introduce caching or a distributed database setup each coming with added complexity.

II.2.3.1 Enforcing Control

With all the pieces together, we can discuss how the server takes this model and enforces the desired access levels on a per user-robot relationship. Whenever the server receives a request from the client to perform a remote procedure call, or in other words invoke any of the RoboScape commands, it looks up details about the robot from its data store. This data can be retrieved from multiple places, more on this later.

If the received request is targeting specific robots, the server first retrieves information about that particular robot or robots and starts by checking their access mode. If the robot(s) are set to public access mode then no further access control is needed and the request is authorized and is passed off to the proper handler. On the other hand, when the server finds out that the access mode for the robot is not public it proceeds to look up information about the caller to verify their access to the robot. Once those pieces of information are gathered and available, if the robot owner has indicated that they want the robot in protected mode, the server then looks at the associated users with the robot to check if the authenticated user has access to it or not. Unauthenticated users (guests) are denied access with an appropriate message and status immediately.

II.2.4 Access Management - UI

Classroom management can be a tricky task, especially when adopting a new curriculum. As part of making RoboScape adoption easier, we make sure that the tools and procedures required to achieve educational goals are intuitive to use and reliable. As discussed, controlling access to the robots is very desirable for a classroom but in order for it to be regularly used, it needs to be easily manageable and that is where a web-based user interface can be very helpful. We created the Teacher Dashboard to support classroom management tools for the instructors including access management control.

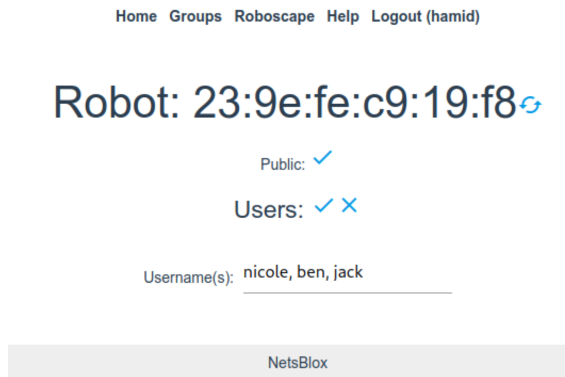

Figure II.8: Individually adding users for controlled access.

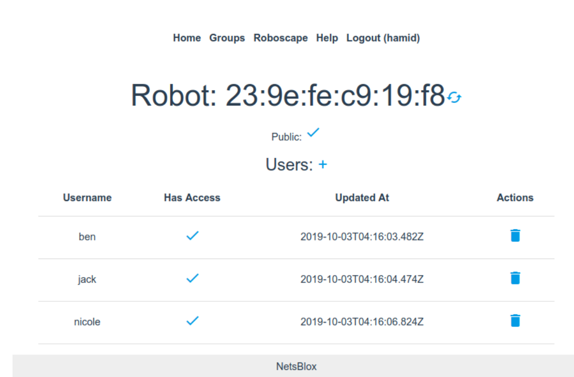

Figure II.9: Controlling access on a per robot basis.

Figure II.10: Adding and viewing user access.

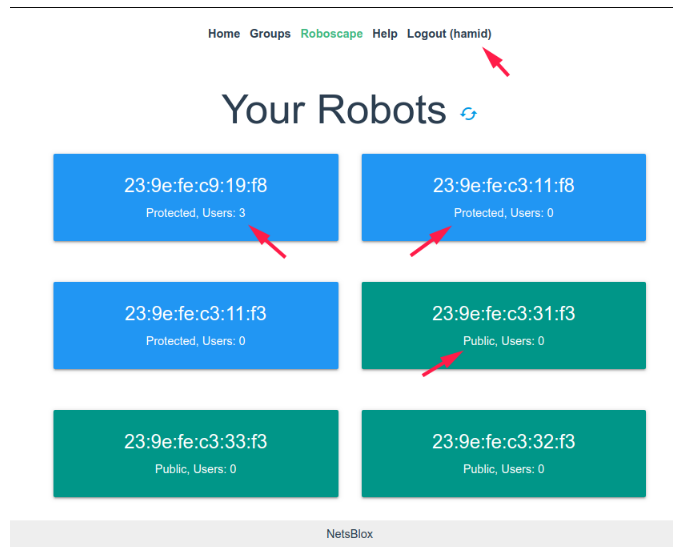

Figure II.11: Robots under the user's control.

In order to support adding and managing users through a web platform, we picked Vue.js [You, 2014] as the frontend JavaScript framework, and extended the existing NetsBlox backend to support the operations

needed by the UI. Vue.js is a progressive JavaScript framework designed for creating interactive web applications with a component-based design focusing on reusability and performance. Figures II.10 and II.11 show some of the robot management views. RoboScape section of this dashboard allows the teacher to:

- View the robots under their ownership.
- Assign users to each robot.
- Revoke access for specific users.
- Change the security mode of each robot.

CHAPTER III

Visual RL

Nowadays we hear new stories about smart homes, self-driving cars, autonomous drones, and many other applications of Artificial Intelligence (AI) every day. Your smartphone uses AI to improve your daily drive to work, giving you better route predictions based on live traffic and city provided information as well as your own driving behaviors and history. Every time you get automated responses to your voice inquires over the phone or through your smart speaker, multiple AI models are coming together to understand and craft the correct response and perform the appropriate actions to your request. Artificial intelligence is changing our lives for the better and in order for us to advance the field, or at the very least be knowledgeable users, we need to understand how they work and what they are about. In some sense, AI may be the most important tool invented by humans. At some point, the capabilities of computational intelligence are expected to surpass humans to an extent that the AI agents improve their own capabilities without any human intervention.

Despite all this potential for good and a great job market, we do not have enough students going into the field: The theories behind AI are very abstract and hard for high school students to get a grasp of. Teaching and learning these concepts to an audience without a strong CS and math background is not trivial nor very interesting. We propose that by using the correct abstractions and picking a select few concepts in AI, we can give students a good introduction to the fundamental concepts at a younger age with less expertise required. In order to have more people in this highly in-demand field of computer science, we need more people to study in this area. But with a high barrier to entry and very theoretical nature of artificial intelligence showing and transferring even the basic ideas can be a challenging task. The question we asked ourselves is can we make AI interesting in a way that is also understandable and approachable? What is a good starting place? How do we approach teaching these concepts?

Artificial intelligence is a very broad umbrella term covering many different forms of deriving intelligence from data. Neural Networks: brain modeling, time series prediction, classification; Evolutionary Computation: genetic algorithms, genetic programming; Vision: object recognition, image understanding; Robotics: intelligent control, autonomous exploration; Expert Systems: decision support systems, teaching systems; Speech Processing: speech recognition and production; Natural Language Processing: machine translation, generating text; Planning: scheduling, game playing; Machine Learning: reinforcement learning, decision tree learning, version space learning.

Some areas of AI are more intuitive than others. Reinforcement learning is based on very intuitive concepts drawing from psychology. This domain focuses on how machines learn to act in a particular way. The

agent consistently learns through associating its actions in an environment with rewards received back from it. These rewards could be immediate or delayed which can make recognizing this association harder for the agent. In all these cases, a lot of the problem design and solutions pull from psychology which makes it closer to everyday life and easier to understand. These similarities can make RL a suitable target as a starting point in introducing AI. As a plus, RL algorithms lend themselves very nicely to game applications which are often the favorite type of programs for young learners.

III.1 Background

Reinforcement learning is a sub-field of machine learning in which computational agents improve their performance by interacting with their environment. Agents exist in an environment with a defined action set and attempt to find an optimal policy that leads them to maximize the reinforcement they receive over the long run e.g., an episode of a game. This policy is a probability distribution over the different actions available to the agent at any given time. Upon taking an action, the agent receives reinforcement signals from the environment to determine how good or bad the action is that they have taken. Sometimes these signals are delayed, for example in a game of tic-tac-toe, the agent does not receive any rewards until the game is over. Reinforcement learning algorithms are very natural to apply to domains such as video games since the rules for states and actions are usually very well defined for these environments. This makes it simpler to implement reinforcement-learning methods in video games as opposed to real-world situations which may have highly unpredictable conditions, continuous state spaces, or inconsistent rules. Additionally, video games are a popular choice of project for students, so if they could be taught some basic reinforcement learning concepts to add to their game projects, it could be a powerful way of introducing artificial intelligence to them.

We introduced reinforcement learning in NetsBlox [NetsBlox, 2016] that is geared toward beginner-level programmers who may not have used any programming language before. This allows users to experiment with reinforcement learning without having to dig deep into programming semantics or all the theories and math behind artificial intelligence. In short, we want to see if we can make teaching (and learning) of AI simpler and spark interest in the field for novice programmers.

III.1.1 Related Work

Teaching AI to young adults is not a new effort. AI for K-12 [Gardner-McCune et al., 2019], ReadyAI, and AI4ALL, among others, are organizations pushing for improving AI education for students of all ages providing researchers with funding and a venue to share their work and accelerate advancement in this area. We looked at different approaches to teaching AI in the field, especially in the context of visual programming and none are utilizing reinforcement learning. Most importantly many previous attempts focus on exposing

students to cloud-based pre-built services. Services such as IBM Bluemix, Azure AI platform, AWS AI services. These products offer machine learning services as a black box where a set API is defined for the user to consume without needing any machine learning knowledge. They include tasks such as media captioning, forecasting, translation, text to speech, picture classification, chatbot services and more. These can be useful in showing what AI can help you do and also as a way of adding advance intelligent to basic projects. But what they lack is the expressive power and learning opportunities that come with building your own AI.

We describe some of the existing platforms working in this area below. Calypso for Cozmo is a robot intelligence framework mixing AI education and robot programming. It incorporates multiple Artificial Intelligence (AI) technologies such as computer vision; face recognition; speech recognition; landmark-based navigation; path planning, and object manipulation. The platform also utilizes rule-based pattern matching language through Microsoft's Kodu Game Lab while also teaching computational thinking via "Laws of Calypso" [Touretzky and Gardner-McCune, 2018].

eCraft2Learn [Kahn and Winters, 2017] is another platform for exposing cloud-based Artificial Intelligence (AI) services in a child-friendly interface powered by Snap!. It requires the user to provide their own Application Program Interface (API) keys from cloud AI providers such as Google, Microsoft, or IBM Watson to power the provided services. Cognimates [Druga et al., 2018] is a project supported by Personal Robots Group at MIT Media Lab where they use Scratch as their visual programming environment of choice for creating kid-friendly interfaces to black-box AI models with a focus on Natural Language Processing (NLP). They provide pre-trained models as well as interfaces for students to feed their own data and retrain their own models.

Last but not least, is Machine Learning For Kids [Lane, 2019] which lets users collect examples of what they want their program to be able to recognize and then provide examples to train the computer to categorize them and finally be able to use this new ability in Scratch to say make a game. They provide an interface separate to Scratch for collecting data and training the model relying on IBM's Watson APIs to power the model.

III.2 Design and Implementation

In this project, we set out to design and develop a sensible framework with the help of a supportive programming environment designed for simplifying complex computing concepts. With appropriate abstractions, smart defaults, and flexible architecture, our goal is to make understanding and implementing some key reinforcement learning algorithms and techniques easy for novice users.

We are implementing a generalized version of procedural Q-learning which is an off-policy, model-free,

temporal difference, reinforcement learning algorithm. In off-policy algorithms, the agent learns from actions that are outside the current policy, such as taking random actions. In this class of algorithms, the agent and our learning process, do not need a policy and a strict set model of the environment which makes getting started with it much quicker. In Q-learning, the agent is optimizing for maximizing the total reward and that. Meaning when the agent is picking actions based on the Q-values in the Q-table, it is considering the future rewards that it expects to receive from taking that action as well as the immediate rewards. In practice, this is reflected in the Q-table and the calculation of the Q-value takes the future reward into account. Every time the agent takes an action a, on an environment with state s, it then observes the new state s', and the reinforcement signal r, it proceeds to update it's Q-table in the following way: 1. Find the best, greediest, next action, a', from the s'. 2. Lookup the maximum Q-value, q', that can be achieved by taking a'. 3. Discount the future q' and add it to r, 4. Update the Q-value for Q(s,a) by a fraction of the difference.

```
Initialize Q(s, a) arbitrarily
Repeat (for each episode):
   Initialize s
   Repeat (for each step of episode):
      Choose a from s using policy derived from Q
         (e.g., ε-greedy)
      Take action a, observe r, s'
      Q(s, a) <-- Q(s, a) + α [r + γ max_α,Q(s', a') - Q(s, a)]
      s <-- s';
   until s is terminal
```

Figure III.1: Q-learning procedural algorithm.

Figure III.1 describes the procedural Q-learning algorithm. γ is the discount factor and α denotes the learning rate. Discount factor often denoted with γ, determines how much reward into the future affects the current actions of the agent or the other way around, how much the agent should value future estimated rewards when picking its current action. Simply put, the discount factor helps balance immediate versus future rewards. In the current version of the Visual RL, this parameter is automatically internally. Learning rate, on the other hand, is a common hyperparameter in machine learning which controls how much each individual training points affect the model. It directly controls how sensitive the training is to noise and stochasticity, and how fast learning happens. It can also be translated into English as the following steps:

1. Initialize the Q-table used for keeping tabs on Q-values for each state-action pair, Q(s, a).
2. Observe the current state, s.
3. Choose an action, a, for state s based on an action selection policy, for example, a greedy policy.
4. Apply the action, and observe the reward, r, as well as the new state, s'.

5. Update the Q-value for the state using the observed reward and the maximum reward possible for the next state.

6. Set the state to the new state, and repeat the process until a terminal state is reached.

Initialized

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	327	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	0	0	0	0	0	0

Training

Q-Table		Actions					
		South (0)	North (1)	East (2)	West (3)	Pickup (4)	Dropoff (5)
States	0	0	0	0	0	0	0
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	328	-2.30108105	-1.97092096	-2.30357004	-2.20591839	-10.3607344	-8.5583017
	⋮	⋮	⋮	⋮	⋮	⋮	⋮
	499	9.96984239	4.02706992	12.96022777	29	3.32877873	3.38230603

Figure III.2: Q-table. Source [LearnDataSci, 2019].

The Q-learning algorithm keeps the action state combinations in a theoretical table (Figure III.2) called Q-table where there are values associated with each state-action pair. These values are usually initialized to zero at the beginning of a new learning process. In a simple implementation, with each reward, the agent looks back at the actions and the states it took them in and updates the Q-value for that state-action pair in its Q-table with a discounted value. Q-learning can be applied to many different situations and some of them can include more than one agent learning different or the same policy. Multi-agent learning could apply to problems where the agent needs to interact with others in an otherwise static environment or when it is playing against itself, called self-play.

Q-learning is a flexible RL algorithm suitable for discrete environments, both deterministic and stochastic. Q-learning's performance can suffer from a big problem space. However, there are techniques and improvements that help in those situations. These techniques include function approximation and quantization to name a few. Quantization looks at reducing action or state space in a meaningful way while capturing most of the expressiveness and the information required for learning. For example, in a simplified game of Snake, instead of encoding the position of the apple as using an (x, y) coordinate, we can encode it as a relative direc-

tion and distance from the head of the snake to reduce the state. Function approximation also plays a big role in making Q-learning a feasible solution for problems with bigger action/state space. Often, neural networks are used as the approximator to enable the agent to estimate values for previously unseen situations and speed up learning. Considering Visual RL is designed for educational purposes in an introductory AI class and in a block-based programming environment, we target problems with smaller action spaces that keep the process simple and more suitable for novice learners. Our implementation and the proposed framework supports multi-agent training with or without policy sharing.

In all cases, the agent is solving an optimization problem to maximize the overall rewards received in the long run. The process of calculating the rewards and giving the appropriate signal to the agent depends upon the task at hand and is going to be part of the environment including, if any, the other agents. Whether the agent is avoiding a predator, seeking a goal state, accumulating treasures, surviving, balancing on a pole, etc, it relies on the environment to provide it with timely feedback to be able to update values on its Q-table. Visual RL provides a set of blocks that facilitate the Q-learning algorithm while being transparent in how they work. It gives control to the user in how they want to set up their project and the learning process. This is achieved by defining a set of carefully designed hook points:

- Create agent: instantiate and initialize a new agent, while defining the action and state space.
- Pick action: pick an action given a state.
- Update step: updates the Q table with the given experience
- Get & set the logic table: provides access to the underlying Q-table.

These hook points are given to the user in form of blocks presented in Figure III.4 and map to each of the mentioned hook points giving the user control over the training and decision-making process of the Q-learning algorithm, providing maximum flexibility while abstracting unnecessary implementation details. Figure III.3 displays the frameworks interactions within NetsBlox.

When integrating Q-learning in a project we start by defining the state and action space keeping in mind that an intelligently reduced problem space can significantly speed up the learning process, whereas a poorly designed one would hamper learning. Once the action and the state spaces are defined, the agent can set up its own Q-table and is ready for action. Assume that we want to train an agent to learn to find a target cell in a grid-like world or maze. In this game, the agent is in an 8 cell by 8 cell grid world with no knowledge of what is around it. We want the agent to be able to reach the closest goal cell without initially knowing where it is as quickly as possible. We choose this simplified version of a maze game so easy to visualize the states and the Q-table. As a first step, we start by thinking about encoding the state, the action space, and our reward

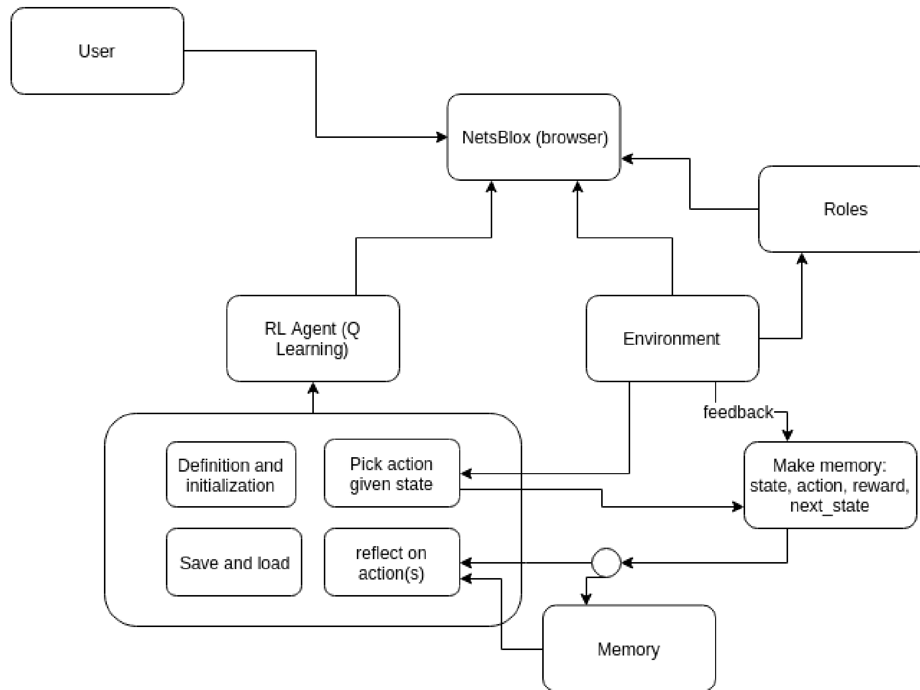

Figure III.3: RL framework interactions within NetsBlox

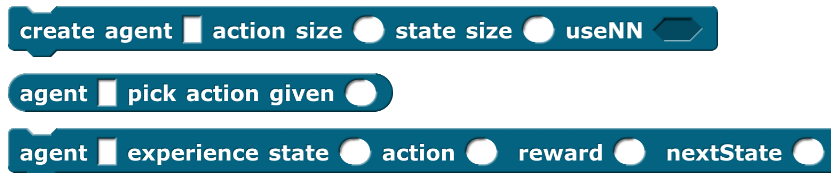

Figure III.4: Core reinforcement learning blocks (Q-learning)

function. Ideally, we want the state and the action space to be as small as possible while allowing them to be expressive enough and give the agent as much relevant information to the task as possible. We define the state as the cells in the map and the actions as four directions: up, down, right, and left. Once we initialize the agent, it internally sets up a Q-table with a height of action size and width of state size to track Q-values for each of these action-state pairs. With these steps all set up, we can start training our agent by just rewarding it the right way and letting it learn from its experiences. With our reward function, we want to enforce the actions that led the agent to the goal state while also discouraging it from moving around for no reason. For example, we can choose to assign a reinforcement of +10 for reaching the end goal and a reinforcement (or reward) of -1 for every step that it takes. Once the game starts, the agent will begin exploring and aim to increase the sum of rewards it is collecting over an episode.

Before we are able to kick off the training, we need to talk about the exploration vs exploitation trade-off. Coming from the exploration vs exploitation trade-off, is the exploration rate. It is the probability of exploring

new possibilities when taking an action given a state versus exploiting what the agent already knows and following its policy. Building upon the grid world example, we can implement a decaying exploration rate so that at the early stages of learning, the agent explores more and as game episodes go on and it has learned the way of the world, it does less exploration and mostly follows greedy actions. More on the grid world example in Section III.3. As with the grid world example, the reward function will always be dependent on the reinforcement learning problem at hand. This function will be completely defined by the user, and the output would be supplied to the RL blocks.

III.2.1 RL Framework and the Blocks

In order to bring RL, and more specifically Q-learning, into the Snap! world, we first needed a flexible in-browser framework for tabular Q-learning. We started by designing and implementing such a framework in the language of the web, JavaScript. The source code for this project can be found on our GitHub page (https://github.com/NetsBlox/Snap--Build-Your-Own-Blocks/tree/rl-snap) or partly in supplemental material A at the end of this document. In designing the underlying JS framework, we set up an agent class called TabularQLearner.

For each agent, a new instance of this class creates the appropriate Q-table based on the given action and state sizes. Each agent exposes methods the following methods:

- Constructor: initial set up of the Q-table, parameters, and the defaults.
- Q-table getters and setters.
- Greedy action policy: determining the highest rewarding action for a given state.
- Update method: Given the state, the action taken, a reward (reinforcement), and the new state, updates the Q-table using the discount factor and the best next Q-value.

In the simple Q-learning setup, the Q-table is implemented as a two-dimensional table, to take direct advantage of native Snap! table visualization and interoperability with Snap! structures. Once this framework was set up and tested we picked all the interesting methods of the agent and exposed it as new blocks in the UI. These blocks, under the hood, create agents using the TabularQLearner class and the utilities around it, to take advantage of the better performance and flexibility of JavaScript compared to Snap! blocks. This also allowed us to further hide unnecessary complexities and provide sane defaults.

Figure III.4 shows the visual representation of these blocks. A potential reinforcement learning scenario for the end user could look like Figure III.6. There are also blocks provided to save and load the agent in the browser as shown in Figure III.5. As well as options to get and set the Q-table that the agent is using under

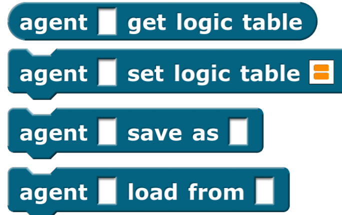

Figure III.5: Utility blocks for saving the agent and working with the Q-table.

the hood for learning and decision making. This is useful in explaining and understanding how the algorithm is working and can also enable shared learning, and transfer learning as well.

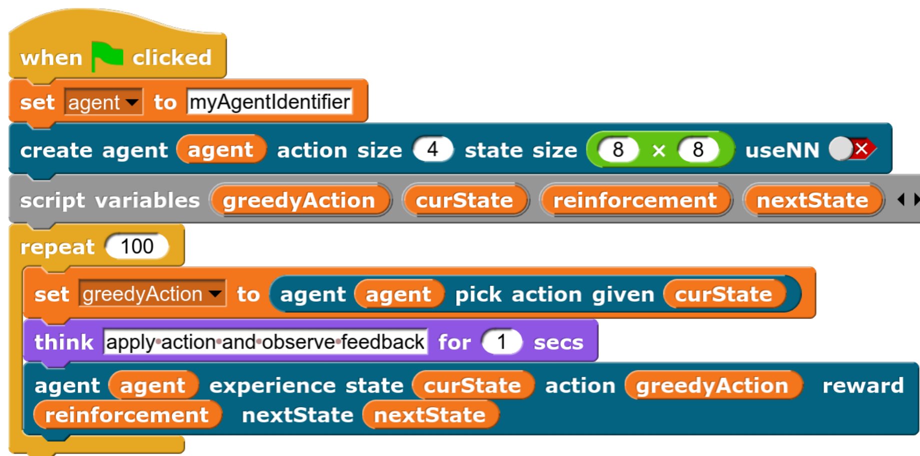

Figure III.6: A basic learning scenario using Visual RL

III.3 Experiments & Results

To demonstrate the framework we describe two different examples. A classic grid world game that requires an intelligent agent, human or AI, to take the shortest path to one of the end goals and a personal pet game in which the user trains a pet (agent) to behave purely by rewarding it.

Following the grid world example from before, we implemented the game environment, agent, the reward function, and everything else needed to showcase the project. Figure III.7 shows a view of the final game environment. A high-level view of the program is displayed in Figure III.8. The project starts by creating the game environment. After the appropriate sprites are added, we create an n-by-n grid with multiple goal states and walls for obstacles. For this example, we pick an 8 by 8 grid. We define a set of three goal cells and enough obstacles to make the routes interesting. We also block off one of the goal states by enclosing it in walls. To add motion to the game, we define a set of helper custom blocks (functions) that when called move the agent (or the player) sprite one unit in the desired direction. We also take into account the obstacles and edges of the map. Once these custom blocks are set up, adding arrow key support is as simple as adding four

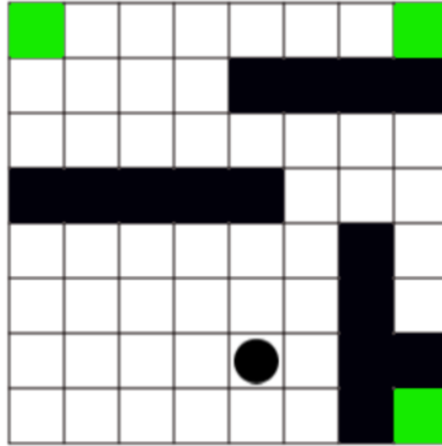

Figure III.7: Grid world game interface

key listeners. The training loop for the agent only needs the underlying movement custom blocks to move around but the key listeners allow a human player to intervene and play the game. Finally, the game state is modeled to correspond to the grid square occupied by the agent at any given time so that each cell can be identified by a single number. As for actions, they are set to represent moving one square up, down, left, or right (assuming the motion is not blocked by the board edge or a wall tile). Depending on the task, inaction could also be encoded as an action which we do not do here.

With the game environment set up, we start working on the training loop. In each iteration, we track counters for steps taken in the episode and the episode (iteration) number. At the start of each game, the agent begins in the starting position of the grid world, 0 in this game, and moves around, learning its policy until it reaches a terminal goal state which ends the episode. The agent always picks the best greedy action that estimates the highest reward gain over an entire episode but since we want the agent to explore we will implement exploration. This means that at every time step when we need an action from the agent, we can instead, with a probability epsilon, pick a random action instead of probing the agent for its choice and use that as selected action. We then give the selected action to the game environment, alongside with the current state of the game if necessary. The environment, the grid world game in this example, processes the inputs and moves to a new state with a reinforcement signal as feedback. In order for the game to come up with the reinforcement signal, it needs a reward function. For this game, we can define a simple reward function such that that actions that put the agent in a non-goal cell have a reinforcement of -1 and actions that land it to a goal states get rewarded with a reinforcement of +10. Figure III.9 shows the block implementation of this reward function.

Once we have the new and the old state, the action that caused the transition, and the reward, we will

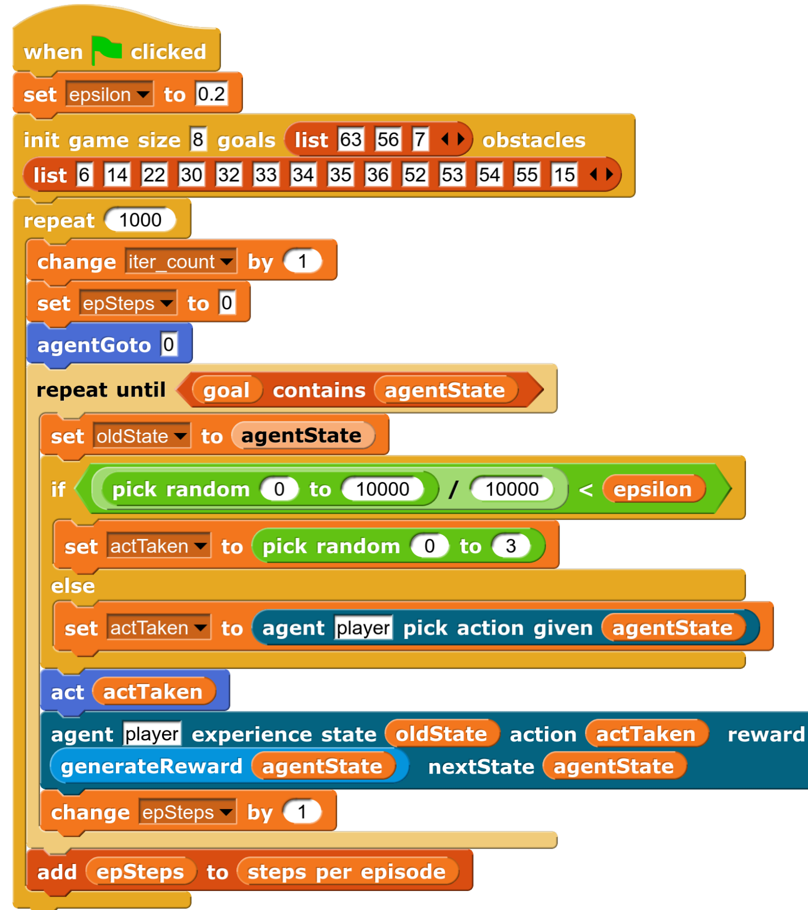

Figure III.8: Building blocks of grid world project.

pass those along to the Q-learner framework as an experience for the agent to train on. We can also keep track of what happened to display training metrics later. These set of steps conclude one step of the training loop in one episode. The game continues until the agent has reached a terminal state at which point the episode ends and another one begins. We set the training to stop after the agent has played a set number of games. However, other stopping conditions could be added such as when the moving average of the steps taken to reach the goal state has not changed for past n episodes. This is similar to early stopping techniques in gradient descent. The agent for this grid world example with a 0.2 epsilon and 0.95 discount rate finds the shortest path to the closest goal in about 150 episodes and consistently reaches the terminal state on the top left corner of the grid with the minimum number of steps. Figure III.10 shows the agent's progress as the game episodes go by. For this experiment, the agent is never told to fully stop exploring and that is visible in the step count variations even after it has found the optimal path. There are a lot more that can be done and taught with a simple example such as the grid world. Some ideas include:

- Saving and loading the logic (Q-table) and sharing it with another agent.

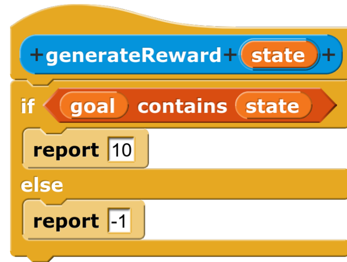

Figure III.9: Grid world reward function.

- Inspecting the logic (Q-table)
- Designing the reward function to reward based on how close the agent is to the goal state.
- Random start position.
- Mixed play between the user and the agent or collaborative play.
- Different algorithms for decaying exploration rate.

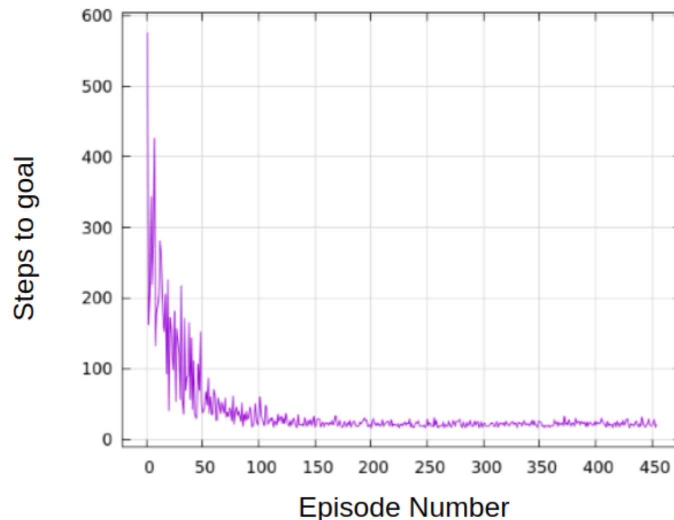

Figure III.10: Number of steps taken by the grid world agent per episode over the course of learning.

In another experiment, we create a pet training game. In this game, the user trains a virtual pet to behave however they like. The idea is that through this exercise the students get a chance to manually control the reward function one by one for the pet. The training happens much more slowly, giving the users the opportunity to study each change separately. Since the choices are made one by one it's easy to change how you are rewarding the pet on the fly to experiment with the effects of, for example, a stochastic reward function or see how different exploration rates could affect learning at different stages.

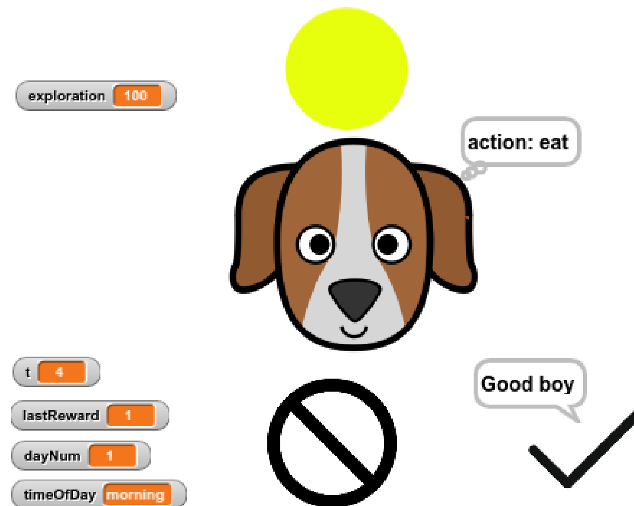

Figure III.11: Pet trainer interface

To create this game, as seen in Figure III.11, we first set up the game environment and mechanics. We define the state space here as a series of discrete timesteps over the course of a “day”: dawn, noon, dusk, midnight, etc. The agent, our pet, can take actions such as jumping around, sleeping, eating, barking, etc. After setting up the graphics, the reward function is created based on user input such that once the game starts the user observes each action of the agent and the state of the game and decides to either praise or criticize the pet, reinforcing the behavior with a positive or negative reinforcement, respectively. The main training loop iterates over a ”day” in the life of this pet starting at early morning and ending at midnight as long as the user wants to continue training their pet. To simplify the time state, as mentioned, it is separated into discrete steps. We designed the time representation this way to lower the state space and make the human-driven reward function possible. An example that needs a lot of training points would be very cumbersome for the user to manually train each action/state pair. It is important to note that the agent will only reliably converge to an optimal policy if the user is consistent in the way they give reinforcements to the agent. This experiment could be a good opportunity for teaching the students about stochastic environments/reward functions, should the instructor choose to go that route in their class.

When we set out to complete this project, we expected to have reinforcement learning functionality implemented in NetsBlox such that a user can easily incorporate RL into their projects and understand the core concepts without having to know much about the unnecessary details and complexity of RL algorithms. Although we have not had a chance to run studies with the platform we would like to include that as a part of future work.

III.4 Future Work

Future work for this project can include adding functionality to support different ways of learning and storing knowledge for the agent(s). For example, we have already included a Boolean switch on the "Create Agent" block to allow the user to toggle between a tabular and deep neural network approach for calculating state/action value updates. We can also roll out the platform to introductory CS courses and survey to see how effective it can be to introducing machine learning to different audiences.

We have yet to conduct any human studies with beginner programmers to see how easy it is for them to incorporate RL into their programs and games (given enough introduction to NetsBlox itself). However, it was relatively simple for us to create an example application and incorporate the Q-Learning tabular method into the example successfully. Given that the added RL blocks only really require the use of about three new constructs, it seems very feasible that given a robust curriculum around Visual RL, students learn about AI and start using RL in their applications.

III.5 Conclusion

Our goal with this project was to bring an understanding of (and interest in) artificial intelligence in general, and reinforcement learning in particular, to a wider audience by making it accessible to those without much in-depth knowledge of computer science concepts. Some initial evaluation of the results we were able to generate shows that our framework will reliably solve some toy problems, and we can extrapolate from that that it will work for any general Markov Decision Processes (MDP) that students can come up with.

Designing this framework was not without its challenges. Coming up with the right level abstractions and the language used proved to be most difficult. Abstracting away unnecessary complexities is helpful to teaching but too much abstraction can cripple the teaching and learning dynamics since the users would not get a good understanding of why they are doing what they doing or how the blocks they are using works. In addition, as it is currently implemented today and the inherent ecosystem of the browser limits how fast and efficient we can do training in this visual environment which would introduce a challenge when the agent is learning higher state spaces.

CHAPTER IV

NetsBlox Player

Smartphone usage and ownership is growing rapidly and has already surpassed PCs and laptops (Figure IV.1). This gap is even bigger in developing countries. Consequently, education platforms that want to be accessible to as many people as possible cannot ignore the dominance of smartphones as today's ubiquitous computing platform.

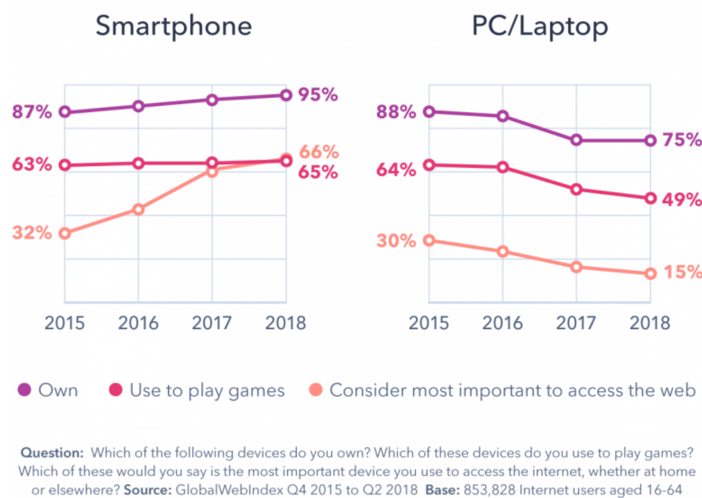

Figure IV.1: Smartphone usage growth versus PC and laptop.

Although NetsBlox is a browser-based application and to some degree adaptive to its container window, the UI is not designed to be used on a small screen device such as a phone or a small tablet. On devices with smaller screens, working with the blocks or dragging and dropping required to build or edit projects becomes difficult to almost impossible. To make matters worse, there is no mouse or physical keyboard on tablets or smartphones. The lack of virtual phone keyboard support prevents the use of any key listeners, key handlers, or text input fields. The main user interface is designed to display multiple different components on it at the same time: the stage, the script area, sprites area, the blocks panel, menus, and more. On a small screen, there is simply not enough real estate to accommodate all of those components at the same time without some major changes. On these devices, project life cycle controls such as project start and stop become hard to operate, dialog boxes for responding to invites or room operations hard to work with, and opening or closing projects through the cloud menu would not be an easy feat either. Part of this inflexibility comes from the Canvas roots of the UI framework used in Snap!. The framework lacks good support for small screen devices

and since the UI is not built using standard Hyper Text Transfer Protocol (HTML) elements and Cascading Style Sheets (CSS) the browsers cannot do much in terms of scaling and reordering the elements properly. Add all these issues together and it makes editing or executing the projects on a small screen impossible. However, with a handful of careful modifications and additions, we can make running NetsBlox and Snap! projects on smartphones and tablets easy and pleasing. Figure IV.2 shows an example of what a project looks like on a small screen.

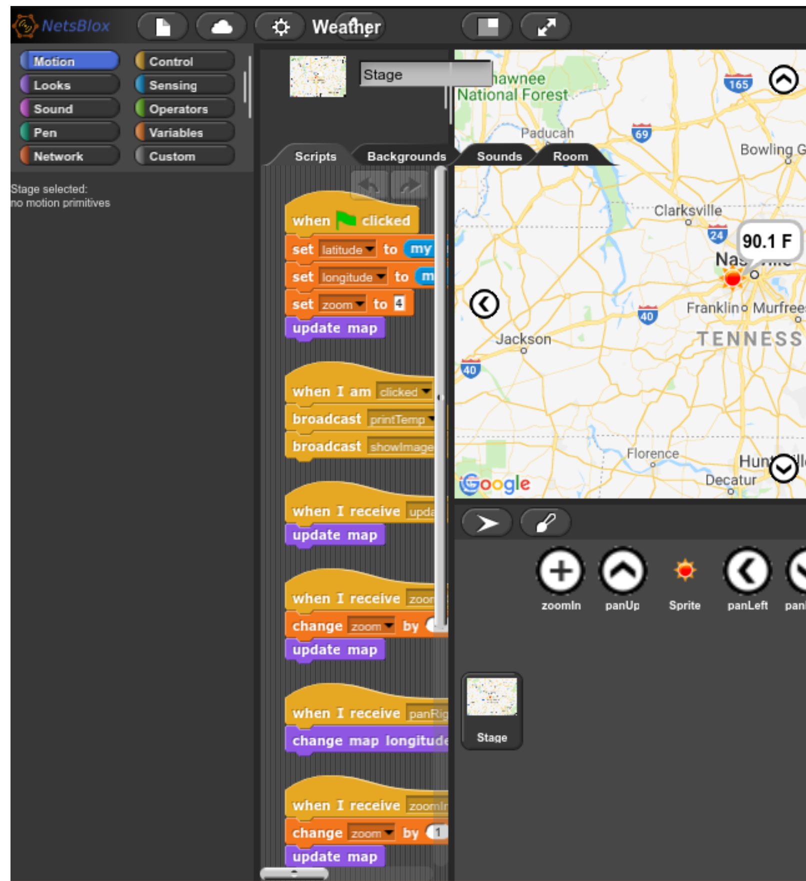

Figure IV.2: Tablet or small desktop screen

IV.1 Methodologies and Implementations

To make running projects on smartphones a reality, without having to ask users to make any changes to their projects, we needed to make multiple updates to NetsBlox client code and provide new features such as interacting with phones' virtual keyboards. We picked smartphone-based applications as the carrier to bundle these modifications and additions as a wrapper around the base client code. In this section, we explain these changes and additions in detail.

Smartphones and tablets run different versions of different operating systems such as Apple's IOS, An-

droid, and Windows Phone, each one having their own Application Program Interface (API) to control different parts of the phone including access to its hardware. Using Apache Cordova [Cordova, 2009], we get a web engine alongside uniform access through standard and user-added APIs to some OS provided features such as controlling the virtual keyboard, native dialogues, persistent storage, network status, and more, To provide a native mobile-looking visual experience we use the Ionic framework [Wilken, 2015]. An interface built on top of Ionic reacts to the platform it is running on to provide a native application experience to the user. This means applying different styles for IOS, Android, Windows Phone, etc.

Cordova and Ionic together also provide the necessary abstraction layers to allows us to write an application once in JavaScript and run it on multiple platforms while ensuring that we can still operate phone features the same way. Ionic after detecting the platform it is running on adapts the UI and the prompts so that it looks much like a native application in whichever platform it is running on, be it Android, or IOS.

We set up the user interface around basic functionalities needed to browse, open, and execute a project. These include authentication, server selection, browsing examples, listing and searching user projects, and storing login information among others. The main challenging part comes after the user is logged in and is ready to launch their project. Each project depends on libraries and the run-time environment provided by the NetsBlox Editor, or Snap! in case of pure Snap! projects, to run. Each NetsBlox project includes: project metadata including project name, NetsBlox version, description (note); one or more roles; block positions and parameters; media as in costumes and audio; and finally Sprites, their positions, and their metadata.

At build time, the application pulls in the latest NetsBlox client code and patches it with the necessary modification and features to complement the UI and improve performance. Some of these modifications are:

- Use the phone's Global Positioning System (GPS) to complement the location block. This requires the user's consent to access location information.
- Redirect NetsBlox notifications and transform netsblox notifications to native phone alerts.
- Tie the Editor's virtual keyboard triggers to the phone's virtual keyboard.
- Simplify the touch mode detection logic.
- Graceful exception handling with retry option.
- Redesign the native presentation (app) mode.

IV.2 Authentication & Projects

The first step of running a project is to find it, and before we can retrieve and show user projects, the application needs to know more about the user and which server it wants to talk to. Since logging is not possible with

the native UI editor's UI we designed a responsive login page in the application that takes the user credentials and passes them on to the NetsBlox server for verification. Once the user has successfully logged in, the application sets and remembers the appropriate cookies on the client to keep the user logged in throughout their use of the application until they explicitly log out. With the user signed in, the application is authorized to access their list of projects. For each project, it receives metadata that includes a thumbnail, project name, and project notes. This information is used to list the projects on a separate tab alongside a thumbnail and a description of the project. Launching a project becomes, for the user, as simple as selecting it and touching "open". This view is shown in Figure IV.3. Tapping the open button launches the bundled and modified NetsBlox client inside an IFrame and opens the selected project. Alongside a list of user projects, a set of curated example projects is also presented to all users to explore without the need for a user account.

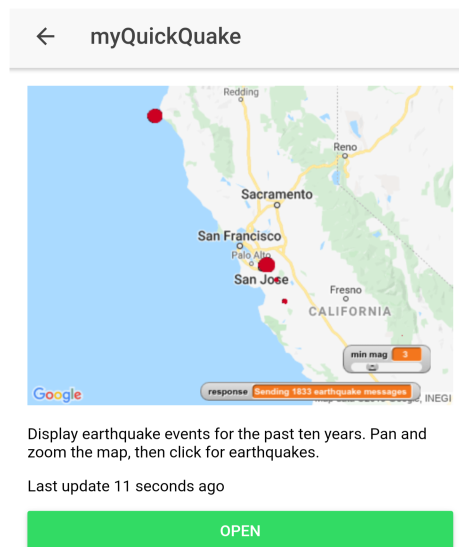

Figure IV.3: Project view.

IV.3 Typing Support

Since the input fields in the NetsBlox editor are not standard HTML inputs it uses a virtual keyboard concept which creates a standard hidden text input element to capture the text. On each keystroke, this text is transferred character by character to the canvas created input field. On smartphones, virtual keyboards do not play well with hidden, out of the view input fields. Having access to the phone APIs, we modified the input fields to trigger the showing of the keyboard. Focusing the canvas input also creates a valid HTML input field directly on top of the original one. The positioning of the hidden input field matters because the Webview engine running the project scrolls the page to bring the focused input into view.

Once the keyboard is up, depending on the phone and orientation, it takes up anywhere between one fifth to half of the screen real estate. In order to accommodate this space and keep the relevant information on the screen, we have to re-size and re-scale the view on the fly. After the input field is submitted the phone is called to hide the keyboard allowing the view to go back to its original full-screen size.

IV.4 Key Listeners

While the typing support IV.3 essentially reads key presses from a virtual (soft) keyboard, it is not suitable for triggering event handlers and key listeners. The soft keyboard takes up a considerable amount of real estate and therefore it cannot be always in display and that makes triggering and hiding it an extra tedious step. Besides, the soft keyboard does not provide some of the most common keys used in user projects such as arrow keys. Also, holding the soft keys will not repeat the keypress which is something some projects can rely on a desktop computer.

To overcome these shortcomings we introduced dynamic keys. Whenever a project is opened it is scanned for its key-listeners. These blocks in NetsBlox, or Snap!, are limited to the following blocks shown in Figure IV.4. Once these key listeners are detected, we create and display soft keys as buttons with hold-to-repeat functionality, saving time and screen space.

Figure IV.4: Key listener blocks

IV.5 Room & Networking

A big part of NetsBlox's offering is its distributed computing and networking capabilities. Most networked projects in NetsBlox use the simulated LAN, or the room concepts, to communicate with people that they invite into their projects. This makes room management an important feature to have on the Player in order to fully support NetsBlox projects. Figure IV.5 shows a view of the room management page and the available operations. We provide means for the users to:

- Accept invitations with mobile native dialog boxes.
- Monitor their network including different roles and their occupants.
- Change into other roles in the project.
- Invite or kick other members to and from a particular role.

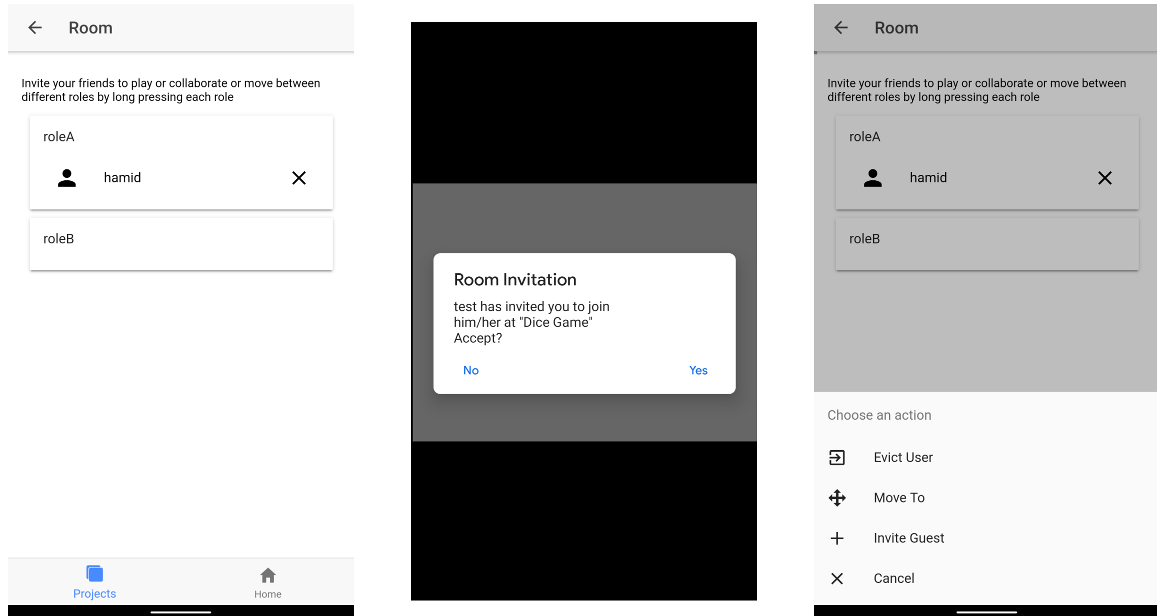

Figure IV.5: Room management in NetsBlox Player.

IV.6 Multiple Platforms & Servers

Thanks to Apache Cordova, our Player application runs on both IOS and Android-powered devices. However, this approach also has its limitations. The two platforms are not built equally and not every feature is supported on both. For example, IOS does not provide an API for opening or closing the virtual keyboard programmatically which limits and degrades its usages in the Player on IOS devices. Also, Cordova on IOS uses WebKit for rendering web pages which is not the recommended web engine by NetsBlox and is considerably slower.

No matter the platform you are running the application on, sometimes you need to change the server it is talking to, be it for development purposes or if you running NetsBlox locally.Changing the server address is achievable by setting this address through the application. Once configured all the application requests will get directed to the desired server.

CHAPTER V

Discussion & Conclusion

There is still a lot that can be done in this area to improve the learning experience of students and teachers. In particular, we would like to run studies with students with different backgrounds to put Visual RL through further testing and conduct studies to help us better understand the strong and weak points of the platform. Considering NetsBlox Player, the option to save and load projects offline, and extended support for the use of native phone sensors come to mind. Furthermore, to improve the user experience on RoboScape, setting up the option to change the target server for the robots, speeding up the batch configuration process, adding advanced triggers for entering configuration mode, and captive portal detection and resolution are good next steps.

Cybersecurity and AI are abstract concepts and understanding such topics has always been challenging for students, especially if they do not have a good CS background. However, in this modern era where we are always connected to the Internet and dependent on computer systems more than ever, it is important to introduce these topics to students and help them understand their importance and impact on society. Enabling hands-on education of Cybersecurity and artificial intelligence is a good step toward achieving this goal.

We have run multiple successful studies using RoboScape and NetsBlox with students ranging from middle school all the way up to K-12, and first year of college in the case of NetsBlox. Visual programming environments and specifically block-based languages are well-positioned to help in driving CS education forward. Realizing the impact of NetsBlox and RoboScape, the unique set of contributions presented in this work cover different aspects of classroom management by providing a teacher dashboard, asset provisioning tools, and access control. In addition, we introduced Visual RL as an intuitive framework for creating and training AI for simple games in the browser using a block-based language. Last but not least, we showed how NetsBlox Player can help make students' projects more interesting by providing an optimal way for the users to execute their games and application on mobile platforms.

BIBLIOGRAPHY

ABI (2016). Abi research anticipates accelerated adoption of automotive software.

Broll, B., Lédeczi, Á., Zare, H., Do, D. N., Sallai, J., Völgyesi, P., Maróti, M., Brown, L., and Vanags, C. (2018). A visual programming environment for introducing distributed computing to secondary education. *Journal of Parallel and Distributed Computing*, 118:189–200.

Cordova (2009). Cordova: Open Source Cross-platform Mobile Application Framework. https://cordova.apache.org. Cited 2015 November 1.

Druga, S., Vu, S., Likhith, E., Oh, L., Qui, T., and Breazeal, C. (2018). Cognimates.

Espressif (2018). Esp touch.

Fraser, N. (2015). Ten things we've learned from blockly. In *Blocks and Beyond Workshop (Blocks and Beyond), 2015 IEEE*, pages 49–50. IEEE.

Gardner-McCune, C., Touretzky, D., Martin, F., and Seehorn, D. (2019). Ai for k-12: Making room for ai in k-12 cs curricula. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pages 1244–1244. ACM.

Grover, S. and Pea, R. (2013). Computational thinking in k–12: A review of the state of the field. *Educational Researcher*, 42(1):38–43.

Harvey, B., Garcia, D. D., Barnes, T., Titterton, N., Miller, O., Armendariz, D., McKinsey, J., Machardy, Z., Lemon, E., Morris, S., et al. (2014). Snap!(build your own blocks). In *Proceedings of the 45th ACM technical symposium on Computer science education*, pages 749–749. ACM.

Instruments, T. (2017). Cc3200 smartconfig provisioning.

Intel (2019). Intel iot overview.

Kafai, Y. B. and Burke, Q. (2013). Computer programming goes back to school. *Phi Delta Kappan*, 95(1):61–65.

Kahn, K. and Winters, N. (2017). Child-friendly programming interfaces to ai cloud services. In *European Conference on Technology Enhanced Learning*, pages 566–570. Springer.

Lane, D. (2019). Machine learning for kids.

LearnDataSci (2019). Q-table.

Margolis, J., Fisher, A., and Miller, F. (2000). The anatomy of interest: Women in undergraduate computer science. *Women's Studies Quarterly*, 28(1/2):104–127.

Morgan, S. (2019). Cybersecurity ventures.

N/A (2000). Digi Xbee S68. https://www.digi.com/products/embedded-systems/rf-modules/2-4-ghz-modules/xbee-wi-fi. Cited 2019 Sep.

NetsBlox (2016). NetsBlox. https://netsblox.org. Cited 2018 August.

Parallax (2019).

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., Millner, A., Rosenbaum, E., Silver, J., Silverman, B., et al. (2009). Scratch: programming for all. *Communications of the ACM*, 52(11):60–67.

Schwab, K. (2015). The future of iot.

Schwab, K. (2018). Future of jobs.

Stamford (2018). Worldwide wearable device sales to grow 26 percent in 2019.

Touretzky, D. S. and Gardner-McCune, C. (2018). Calypso for cozmo: Robotic ai for everyone. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, pages 1110–1110. ACM.

Viehböck, S. (2011). Brute forcing wi-fi protected setup. *Wi-Fi Protected Setup*, 9.

Viswanathan, P., Batra, V., and Vyas, P. (2015). Convenient use of push button mode of wps (wi-fi protected setup) for provisioning wireless devices. US Patent 9,191,771.

Wang, X. (2013). Why students choose stem majors: Motivation, high school learning, and postsecondary context of support. *American Educational Research Journal*, 50(5):1081–1121.

Wilken, J. (2015). Ionic. https://ionicframework.com/. Cited 2019 August.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3):33–35.

Wolber, D., Abelson, H., Spertus, E., and Looney, L. (2011). *App Inventor*. " O'Reilly Media, Inc.".

You, E. (2014). Vue.js. https://vuejs.org. Cited 2019 August.

Appendix A

Q Learner Framework

src/tabularRl.js

```
require=(function(){function r(e,n,t){function o(i,f){if(!n[i]){if(!e[i]){var c
    ="function"==typeof require&&require;if(!f&&c)return c(i,!0);if(u)return u(i
    ,!0);var a=new Error("Cannot find module '"+i+"'");throw a.code="
    MODULE_NOT_FOUND",a}var p=n[i]={exports:{}};e[i][0].call(p.exports,function(
    r){var n=e[i][1][r];return o(n||r)},p,p.exports,r,e,n,t)}return n[i].exports
    }for(var u="function"==typeof require&&require,i=0;i<t.length;i++)o(t[i]);
    return o}return r})()({1:[function(require,module,exports){
const utils = {
  /**
   * Shuffles array in place. ES6 version
   * @param {Array} a items An array containing the items.
   */
  shuffle(a) {
    for (let i = a.length - 1; i > 0; i--) {
      const j = Math.floor(Math.random() * (i + 1));
      [a[i], a[j]] = [a[j], a[i]];
    }
    return a;
  },

  getRandomInt(max) {
    return Math.floor(Math.random() * Math.floor(max));
  },

  pickRandom(list) {
    let idx = this.getRandomInt(list.length - 1);
    return list[idx];
  },

  sampleBatch(arr, batchSize) {
    console.assert(arr.length >= batchSize, 'the array is smaller than batchSize
        ');
    let maxBatchStart = arr.length - batchSize;
    let batchStartIndex = this.getRandomInt(maxBatchStart);
    return arr.slice(batchStartIndex, batchStartIndex + batchSize);
  },

  indexOfMax(arr) {
    if (arr.length === 0) {
      return -1;
    }

    let max = arr[0];
    let maxIndex = 0;

    for (var i = 1; i < arr.length; i++) {
      if (arr[i] > max) {
        maxIndex = i;
        max = arr[i];
      }
    }
```

```
  return maxIndex;
},

download(filename, text) {
  let pom = document.createElement('a');
  pom.setAttribute('href', 'data:text/plain;charset=utf-8,' +
      encodeURIComponent(text));
  pom.setAttribute('download', filename);

  if (document.createEvent) {
    let event = document.createEvent('MouseEvents');
    event.initEvent('click', true, true);
    pom.dispatchEvent(event);
  } else {
    pom.click();
  }
},

// onehot encodes 1d array
onehot(list, depth) {
  let oh = list
    .map(val => {
      let vec = new Array(depth).fill(0);
      vec[val] = 1;
      return vec;
    })
    .reduce((a, b) => a.concat(b));
  return oh;
},

createMemory(capacity) {
  class Memory {
    constructor(size) {
      this._size = size;
      this._storage = [];
    }

    remember(item) {
      this._storage.push(item);
      if (this._storage.length > this._size) {
        this._storage.shift();
      }
    }

    get memories() {
      return this._storage;
    }
  }
  return new Memory(capacity);
},

// checks array equality
areEqual(arr1, arr2) {
  if(arr1.length !== arr2.length)
    return false;
  for(let i = arr1.length; i--;) {
    if(arr1[i] !== arr2[i])
      return false;
  }
```

```
    return true;
  }

};

module.exports = utils;

},{}],"tabularQLAgent":[function(require,module,exports){
const utils = require('./utils');

class TabularQLearner {
  constructor(opts) {
    const defaults = {
      discountRate: 0.95,
      epsilon: 1,
      actionSize: undefined,
      stateSize: undefined,
      updateRate: 0.1,
    };
    opts = {...defaults, ...opts}; // apply the defaults

    console.log('creating tabular qlearning agent', opts);

    // Create qTable and fill with random numbers
    this._qTable = [];
    for (var state = 0; state < opts.stateSize; ++state) {
      let action_array = [];
      for (var action = 0; action < opts.actionSize; ++action) {
        action_array.push(Math.random());
      }
      this._qTable.push(action_array);
    }
    this.state_size = opts.stateSize;
    this.action_size = opts.actionSize;
    this.discount_factor = opts.discountRate; // gamma
    this.update_rate = opts.updateRate; // alpha
    this.epsilon = opts.epsilon;
  }

  // getQ
  q(state, action) {
    return this._qTable[state][action];
  }

  _setQ(state, action, value) {
    this._qTable[state][action] = value;
  }

  // Since the "utils.indexOfMax" function only returns one action
  // we don't need the _best_actions/best_action combo
  // If this isn't "random enough" in the case of a tie,
  // we can modify how indexOfMax handles ties
  best_action(state) {
    if (state >= this.state_size) throw new Error(`Invalid choice of state: ${
        state}. Must be less than state_size ${this.state_size}`);
    return utils.indexOfMax(this._qTable[state]);
  }

  act(state) {
```

```
    let action = null; //null or undefined? Does it matter?
    if (Math.random() <= this.epsilon) {
      action = utils.getRandomInt(this.action_size);
    } else {
      action = this.best_action(state);
      if (action >= this.action_size || action < 0) {
        throw new Error(`Invalid choice of action: ${action}. Must be between 0
            and ${this.action_size - 1} (inclusive)`);
      }
    }
    return action;
  }

  update(state, action, reward, next_state) {
    // compute the difference
    const best_next_action = this.best_action(next_state);
    const best_next_value = this.q(next_state, best_next_action);
    const diff = reward + (this.discount_factor * best_next_value) - this.q(state
        , action);
    // compute the update
    const new_value = this.q(state, action) + this.update_rate * diff;

    // do the update
    this._setQ(state, action, new_value);
  }
}

module.exports = TabularQLearner;

},{"./utils":1}]},{},[]);
```

src/agent.js

```
'use strict';
/* global tf, _, utils */

/* AI architectures */

/* throughtout this code there are conversions between tf tensors and simple
     arrays that could be avoided to improve performace
  */

class Agent {
  constructor(opts) {
    const defaults = {
      discountRate: 0.95,
      epsilon: 1,
      epsilonMin: 0.02,
      epsilonDecay: 0.99,
      model: undefined, // dnn
      targetModel: undefined,
      batchSize: 32,
      actionSize: undefined,
      stateSize: undefined
    };
    opts = _.extend(defaults, opts);
    console.log('creating agent', opts);

    if (!this._checkInitOpts(opts)) throw new Error('Bad agent initialization');
    for (let key in opts) {
```

```
    this[key] = opts[key];
   }

   this.model = this.model || this._createModel();
   this.targetModel = this.targetModel || this._createModel();
  }

  _createModel() {
   // input size is stateSize. output size is actionSize
   const model = tf.sequential();
   model.add(tf.layers.dense({units: 24, activation: 'relu', inputShape: [this.
       stateSize]}));
   model.add(tf.layers.dense({units: 24, activation: 'relu'}));
   model.add(tf.layers.dense({units: this.actionSize, activation: 'linear'}));
   model.compile({loss: 'meanSquaredError', optimizer: 'adam'});
   return model;
  }

  // creates a tensor from the state,
  // this is used to allow the memory to store the state as simple arrays (not
     optimal)
  _makeTensor(state) {
   let stateTensor = tf.tensor2d(state, [1, state.length]);
   return stateTensor;
  }

  async act(state) {
   state = this._makeTensor(state);
   if (Math.random() <= this.epsilon) {
    return utils.getRandomInt(this.actionSize);
   }
   let actVals = await this.model.predict(state).dataSync();
   return utils.indexOfMax(actVals); // returns action
  }

  async replay(memories) {
   if (memories.length < this.batchSize) {
    return false;
   }
   // take a random batch from memories train on it
   // QUESTION shuffle the whole thing take a random batch
   // 1. how manytimes do we shuffle the whole thing?
   // 2. be able to remove old memories
   const minibatch = utils.sampleBatch(memories, this.batchSize); // non
       consecutive

   async function replayMemory(singleMemory) { // FIXME
    let [state, action, reward, nextState, done] = singleMemory;
    let stateTensor = this._makeTensor(state);
    let nextStateTensor = this._makeTensor(nextState);
    let target = await this.model.predict(stateTensor);
    target = target.buffer(); // convert tensor into tensor buffer to
        manipulate
    if (done) {
     target.values[action] = reward;
    } else {
     let a = await this.model.predict(nextStateTensor).data();
     let t = await this.targetModel.predict(nextStateTensor).data();
     target.values[action] = reward + this.discountRate * t[utils.indexOfMax(a)
```

```
        ];
      }
      target = target.toTensor();
      await this.model.fit(stateTensor, target, {
        // batchSize: this.batchSize,
        // epochs: 1
      });
    }

    for (let i = 0; i < minibatch.length; i++) {
      await replayMemory.call(this, minibatch[i]);
    }

    this._decayEpsilon();
  }

  async save(location) {
    print('saving model..', location);
    const saveResults = await this.model.save(location);
    return saveResults;
  }

  async load(location) {
    console.log('loading model..', location);
    this.model = await tf.loadModel(location);
    this.targetModel = await tf.loadModel(location);
    return this.model;
  }

  _checkInitOpts() {
    return true;
  }

  _decayEpsilon() {
    if (this.epsilon > this.epsilonMin) {
      this.epsilon *= this.epsilonDecay;
    }
  }

  async updateTargetModel() {
    // copy weights from model to targetModel
    // here we are copying the whole model. OPTIMIZE?
    const SAVEPATH = 'localstorage://dqnModel';
    const saveResults = await this.model.save(SAVEPATH);
    this.targetModel = await tf.loadModel(SAVEPATH);
  }
}
```

src/drivers/tic.js

```
/* global NewGame, Game, board, utils */
// game driver prepares the game to be used by the RL player

class TicGame extends Game {
  constructor(opts) {
    super(opts);
  }

  reset() {
    console.debug('resetting game');
```

```
  NewGame();
}

// Q set it up as getter?
state() {
  const conversionMap = {};
  conversionMap[UNOCCUPIED] = 0;
  conversionMap[HUMAN_PLAYER] = 1; // player one or the ai
  conversionMap[COMPUTER_PLAYER] = 2;
  let state = board.map(v => {
    return conversionMap[v];
  });

  // onehot
  // let stateTensor = tf.oneHot(tf.tensor1d(state, 'int32'), 3);
  state = utils.onehot(state, 3);
  return state;
}

async act() {

}

feedback() {

}

async step(action, role) {
  let reward, isDone, newState, info;
  reward = 0;
  if (!GetAvailableMoves(board).includes(action)) {
    console.debug('trying to mark an occupied slot');
    newState = this.state();
    reward = -20;
    isDone = true;
    return [newState, reward, isDone, info];
  }
  // prepare the newState
  MakeMove(action);
  newState = this.state();

  // is it finished?
  let curStatus = CheckForWinner(board);
  isDone = curStatus !== 0; // OPTIMIZE recomputing the winner

  // define the rewards
  if (isDone) {
    if (curStatus === 1) { // tie
      reward = 0;
    } else if (curStatus === 2) { // $HUMAN_PLAYER won
      reward = 10;
    } else { // $COMPUTER_PLAYER won
      reward = -10;
    }
    console.debug('rewarded', reward);
  } else { // reward for picking a valid action..
    reward = 1;
  }
  return [newState, reward, isDone, info];
```

```
  }
}
```

src/game.js

```
// game super class

class Game {
  constructor(opts) {
    console.log('creating game driver');
    this._test();
  }

  get state() {
    throw new Error('Method is not implemented.');
  }

  async step(action) {
    throw new Error('Method is not implemented.');
  }

  reset() {
    throw new Error('Method is not implemented.');
  }

  async act(action) {
    throw new Error('Method is not implemented.');
  }

  // run once after an action to get its feedback
  get feedback() {
    throw new Error('Method is not implemented.');
  }

  // basic tests for the implemented game
  _test() {
    console.assert(this.state() instanceof tf.Tensor || Array.isArray(this.state
        ()), 'state is not formatted properly');
  }
}
```

src/player.js

```
/* global _ */

class Player {
  constructor(opts) {
    let defaults = {
      maxMoves: 1000
    };
    opts = _.extend(defaults, opts);
    console.log('configuring player with', opts);
    this.maxMoves = opts.maxMoves;
    this.name = opts.name;
    this.game = opts.game;
    this.agent = opts.agent;
    this.role = opts.role;
    this.memory = new utils.createMemory(opts.memorySize);
  }

  get saveLocation() {
```

```
    return this.name + '-gameName-agent.save';
  }

  async runEpisode(e) {
    this.game.reset();
    let state = this.game.state();
    let cumRewards = 0;
    for (let moveNum = 0; moveNum < this.maxMoves; moveNum++) {
      let action = await this.agent.act(state);
      console.debug(this.name, 'action', action);
      let [nextState, reward, isDone, info] = await this.game.step(action, this.
          role);
      console.debug('after action step', reward);
      cumRewards += reward;
      this.memory.remember([state, action, reward, nextState, isDone]);
      state = nextState;
      if (isDone) {
        console.log(`### finished ep w/ ${moveNum + 1} moves & rewards ${
            cumRewards}`);
        if (this.onEpisodeDone) this.onEpisodeDone({episode: e, cumRewards});
        return await this.agent.replay(this.memory.memories);
      }
    }
    console.log('ran out of moves');
  }

  async train(opts) {
    let defaults = {episodes: 100, updateFreq: 2, saveFreq: undefined};
    opts = _.extend(defaults, opts);
    for (let e = 0; e < opts.episodes; e++) {
      console.log(`### episode ${e}, epsilon ${this.agent.epsilon}`);
      await this.runEpisode(e);
      if (e % opts.updateFreq === 0) {
        await this.agent.updateTargetModel();
      }
      if (opts.saveFreq && (e % opts.saveFreq === 0)) {
        await self.agent.save(this.saveLocation);
      }
    }
  }
}
```

src/players/tic.js

```
/* global TicGame, Player, Agent */

const BOARDSIZE = 3 * 3;
DIFFICULTY = 0.4;
VISUALIZE = false;

NewGame();
let game = new TicGame();
let agent = new Agent({actionSize: BOARDSIZE, stateSize: game.state().length,
    epsilonDecay: 0.99});
let player = new Player({game: game, agent: agent, name: HUMAN_PLAYER + 'Player
    ', maxMoves: 5, role: HUMAN_PLAYER, memorySize: 500});

var ctx = document.getElementById('chart').getContext('2d');
var chart = new Chart(ctx, {
  // The type of chart we want to create
```

```
  type: 'line',

  // The data for our dataset
  data: {
    labels: [1, 2],
    datasets: [{
      label: 'My First dataset',
      // backgroundColor: 'rgb(255, 99, 132)',
      // borderColor: 'rgb(255, 99, 132)',
      data: [-18, -17]
    }]
  },

  // Configuration options go here
  options: {}
});

const rewardsHistory = [];

player.onEpisodeDone = async function(inp) {
  console.log('finished episode with rewards', inp);
  rewardsHistory.push(inp.cumRewards);
  chart.data.labels.push(chart.data.labels.length + 1);
  chart.data.datasets[0].data.push(inp.cumRewards);
  if (inp.e % 100 === 0) {
    chart.update();
    player.agent.save('localstorage://ticmodel');
  }
};

function startTraining() {
  let numEpisodes = parseInt(document.getElementById('numEpisodes').value);
  let startTime = new Date().getTime();
  player.train({episodes: numEpisodes})
    .then(arg => {
      let duration = new Date().getTime() - startTime;
      console.log('finished training in', duration / 1000);
      chart.update();
    })
    .catch(err => {
      console.error(err);
      console.log('there was an error during training');
    });
}
```

src/utils.js

```
const utils = {
  /**
   * Shuffles array in place. ES6 version
   * @param {Array} a items An array containing the items.
   */
  shuffle(a) {
    for (let i = a.length - 1; i > 0; i--) {
      const j = Math.floor(Math.random() * (i + 1));
      [a[i], a[j]] = [a[j], a[i]];
    }
    return a;
  },
```

```
getRandomInt(max) {
  return Math.floor(Math.random() * Math.floor(max));
},

pickRandom(list) {
  let idx = this.getRandomInt(list.length - 1);
  return list[idx];
},

sampleBatch(arr, batchSize) {
  console.assert(arr.length >= batchSize, 'the array is smaller than batchSize
      ');
  let maxBatchStart = arr.length - batchSize;
  let batchStartIndex = this.getRandomInt(maxBatchStart);
  return arr.slice(batchStartIndex, batchStartIndex + batchSize);
},

indexOfMax(arr) {
  if (arr.length === 0) {
    return -1;
  }

  var max = arr[0];
  var maxIndex = 0;

  for (var i = 1; i < arr.length; i++) {
    if (arr[i] > max) {
      maxIndex = i;
      max = arr[i];
    }
  }

  return maxIndex;
},

download(filename, text) {
  var pom = document.createElement('a');
  pom.setAttribute('href', 'data:text/plain;charset=utf-8,' +
      encodeURIComponent(text));
  pom.setAttribute('download', filename);

  if (document.createEvent) {
    var event = document.createEvent('MouseEvents');
    event.initEvent('click', true, true);
    pom.dispatchEvent(event);
  } else {
    pom.click();
  }
},

// onehot encodes 1d array
onehot(list, depth) {
  let oh = list
    .map(val => {
      let vec = new Array(depth).fill(0);
      vec[val] = 1;
      return vec;
    })
    .reduce((a, b) => a.concat(b));
```

```
    return oh;
  },

  createMemory(capacity) {
    class Memory {
      constructor(size) {
        this._size = size;
        this._storage = [];
      }

      remember(item) {
        this._storage.push(item);
        if (this._storage.length > this._size) {
          this._storage.shift();
        }
      }

      get memories() {
        return this._storage;
      }
    }
    return new Memory(capacity);
  }

};
```