Automata

A novel game and interactive educational workshop

By

Tristan Fabes

Supervised by Dr Peter Bennett

Department of Engineering Mathematics
University of Bristol

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science in the Faculty of Engineering.

August 2022

Abstract

Procedural learning is the process of acquiring skills and knowledge through the repeated execution of a task. It offers profound opportunities to be successfully deployed in educational games by embedding the academic learning goals of an exercise within the mechanics of the game. To date, no educational video games have been created that are based on the simple rule sets that generate Cellular Automata, whilst also utilising procedural learning to embed concepts of computation as key learning outcomes. This project therefore sets out to build a React web application that offers an interactive workshop, introducing users to Cellular Automata, and a competitive strategy card game that enables users to apply the knowledge learned in the workshop. The platform is intended to be used in the education of secondary school students and first year Computer Science undergraduates.

The project delivers this through the production of the game *Automata*. The game features fully working single, local and online multi-player game modes, and is at its core based around Conway's Game of Life. The educational workshop section of the application is comprised of self-authored content and has a wide range of interactive examples of the starting configurations used in Game of Life. The workshop enables a framing of the learning objectives that are built-in to the game's mechanics, and the game is then utilised to drive the engagement and motivation of users to further their own understanding of Cellular Automata. User-evaluation showed that both the site and the game were robust, easy to interact with, offered strong foundations for learning about Cellular Automata and that the game provided a challenging yet enjoyable user experience.

This project provides a strong introduction to a publicly available web application which can be utilised effectively within educational settings to support the introduction of programming and computation principles.

Access the website at: www.automata-game.co.uk

Dedication and acknowledgements

Dedicated to my fiancé, Ruth, for patiently allowing me to work on this when I should have been wedding planning.

This project is within the scope of the blanket ethics application 12046, as reviewed by my supervisor Peter Bennett.

Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... DATE: 26/08/22

Table of Contents

List of Tables

Table **Page**

List of Figures

List Of Listings

Chapter 1

Introduction

Procedural learning is defined as the process by which the repetition of an action leads to the development of knowledge and cognitive skills that relate to the action carried out [1]. This project seeks to utilise the notion of procedural learning in video games to improve the user's understanding of Cellular Automata. Procedural learning can be embedded into a game's design by making the core game mechanics frame the desired learning outcomes [2], ensuring that the repetitive tasks that constitute players' actions contribute to the formation of implicit memories that can be used to unconsciously improve future performance through pattern recognition and skill acquisition. Although many subject areas could have been chosen, Cellular Automata were selected because they offer a valuable opportunity for such a game due to their highly visual representation, simple underlying rule set, and direct parallels with computation. Against the background of increasingly complex and realistic game engines, it is argued here that it is the simplicity of more rudimentary games that can offer the best platform for eliciting effective educational outcomes by enabling users to have a greater degree of understanding of the system with which they are interacting and through clearer framing of the key learning outcomes.

1.1 Education, Computation and Video Games

Educational video games have become increasingly accessible and popular in the form of apps for smartphones and tablets. The Common Sense Media survey [3], carried out in the US, found that, in 2020, children aged 8 or below spent on average 23 minutes each day playing games. For children aged between 5 and 8, this figure is 40 minutes. This has been a consistent figure since 2011. A persistent question since the introduction of digital games as pedagogical tools is the effectiveness and impact that they have on learners. O'Neil [4] outlined four key areas to answer this question: diverse approaches to learning; interaction; response to cognitive and affective learning issues; and motivation. These evaluation areas are starkly contrasted to how the successes of many video games are evaluated - through their user-base and monetary gain. O'Neil's review of available educational games from the 15 years prior to publication found a significant potential for learning, but a widespread lack of empirical evidence. The issue of evidence is one that persists today [5]. Kacmaz and Dubé's more recent meta-analysis [6] outlined how the specific rule sets of video games, represented through the game's mechanics, enable a framing of an academic concept that the user is repeatedly exposed to through repetition of in-game actions. The decision-making required to play the game is in itself the learning process. This notion of learning through action is by no means new.

> *No compulsory learning can remain in the soul... In teaching children, train them by a kind of game, and you will be able to see more clearly the natural bent of each* [7].

To be effective and engaging tools for learning, games must have an intrinsic motivational drive [8]. This drive can take the forms of competition, increasing difficulty, information that can be discovered and a degree of perceived randomness inherent in the game's systems. Games that promote learning need to inspire curiosity and develop a sense of mastery in the player. These experiences are increasingly difficult to achieve in the complex systems of open world games. Instead, this paper suggests that more simplistic systems are the foundations upon which these goals can be attained. When teaching the notions of computation through the fundamental ideas of state transitions, Cellular Automata (CA) offer an exciting opportunity to create a system that is easy to understand but takes time and effort to master. The most widely known game that uses CA as its basis is John Conway's *Game of Life* [9]. As a zero player simulation game, the game in its original form lacks the interactivity required to create a competitive experience that builds a sense of intrinsic motivation. Whilst some multiplayer variants have been created, many have failed to be successful in capturing a long term audience, and none to date have attempted to create a system that utilises the educational potential of CA for the benefit of the users.

1.2 Project Aims

This project is focused on the creation of *Automata*, a CA based educational video game embedded in a React web application that also has a self-contained educational workshop teaching the fundamentals of CA. The workshop acts as the introductory point to the project for users, and utilises interactive examples of CA to showcase one-dimensional and two-dimensional formats of CA. The game, produced as a Unity WebGL game, is a strategy card game based on Conway's *Game of Life*. The game uses a Pong-style scoring system, simultaneous turn-based card placement, and a *best of five* rounds victory condition. The cards used in the game represent the different *Game of Life* seeds that have been discovered by mathematicians over the past forty years, and the workshop gives an introduction to the different types of seeds and how they evolve over time. The combination of these two products seeks to create a foundation for understanding how CA systems work, followed by an engaging and competitive game that captures users' attentions and creates the motivation for continuing to play as well as a furthering of their understanding of CA. Ultimately, the workshop and game could be of great value to secondary school learners and first year undergraduates in Computer Science. The main contributions that the project offers are:

- Creation of a highly strategic and rewarding game based on CA, featuring single, local and online multi-player game modes, with in-built seed analysis capabilities.
- An interactive workshop teaching the fundamentals of CA systems, with the entire project, including data collection, being embedded in a React web application.
- Embedded user-evaluation on the usability, educational impact, suitability of content, and strengths and improvement areas of the web application and Unity game.

Chapter 2

Background

This chapter introduces the academic context and motivations for utilising CA in educational tools and settings, before going on to detail the principles and rules that comprise Conway's *Game of Life* (*Life*). It then covers the areas of existing work relating to this project, including online variants of *Life*, educational platforms for learning about *Life*, and the different strategy card games that have been influential in the design process of the current project.

2.1 Cellular Automata in Education

Introductory courses in Computer Science and programming, as are studied by students in their final years of secondary school and by first year undergraduates, must be engaging and motivating to ensure strong learning outcomes [10]. Thomas [11] and Lilly [12] both set out cases for how CA can be utilised to generate creative problem-solving exercises for students, introducing them to finite state-machines and ways of modelling naturally occurring phenomena such as the patterns found on Conus seashells [13] or the Belousov Zhabotinsky reaction [14].

Beyond the development of modelling skills and the real-world applicability of CA, they can be used as accessible examples to introduce programming concepts such as: computing state using algorithms based on current state; data types and data collections; object-oriented approaches; and the interactivity of objects. The highly visual aspect of CA facilitates the ability to describe and explain these concepts without requiring students to engage with the code base that generates them.

Lilly [12] provides the initial motivation for the academic direction that this project has taken:

> *Some computer games are based on the notion of cellular automata and students find them engaging and fascinating... Proper use of the fascination can be employed to motivate students' interests in studying important problems and working out solutions. This could lead to students deciding on a career goal rather than dropping out of school.*

Whilst a more extensive detailing is given in Section 2.4 of how fascination and engagement can be nurtured, through the design of educational games in respect of competition and challenge, it is first important to answer the question of why to focus on CA. A key driver behind this decision was the potential for long-term development and scaling up of the project to meet the educational needs of a wide-range of learners. As Staubitz [15] notes, the difficulty level of the subject matter presented through a CA lens can easily be targeted to novices or advanced

programmers - from counting exercises to Turing complete computation. Becker [16] emphasises the need for programming assignments to extend beyond mundane sorting and manipulation of strings in order to capture the attention and interest of learners, and CA offer an exciting opportunity to create just such exercises. Development of an online workshop that, through interactivity and visual representation, brings to life the real-world applicability of modelling natural phenomena with CA is proposed as a means of delivering upon this idea. By pairing the workshop with a competitive game, based on Conway's *Life*, that allows students to play against each other within a classroom or computer lab, it seeks to capture the fascination described by Lilly [12]. The project's output will be assessed against its ability to achieve positive educational outcomes for users, introducing them to and developing an interest for the complexity that can be produced by the simple systems that comprise CA.

2.2 Conway's Game of Life

Life was first popularised in an article in Scientific American in 1970 [9]. It is a zero-player game, where an initial configuration is set up and the simulation is then run with the purpose being to watch the gradual evolution of the configuration without any further input from the user. It is the most well known of the CA that have been developed since their conceptualisation by John von Neumann in 1951 [17]. All CA, including *Life*, follow a set of key principles [18]:

- They are discrete, and therefore have a finite number of cells that constitute an N-dimensional lattice.
- Each cell exists in a finite number of states that must be equal to or greater than 2.
- The state of the cells change uniformly, with changes following an identical set of rules.
- State changes happen simultaneously for every cell.
- The state of cells are only affected by local factors (i.e. the immediately neighbouring cells), and it is not possible for cells to exert influence at a distance.

In *Life*, the lattice used for the game is a two-dimensional grid of square cells. There are two possible states that cells can be in - either *alive* or *dead*. State changes are determined based on the state of the cell and its 8 immediate surrounding neighbours (known as the *Moore-Neighbourhood* [19]). The rule set that determines the fate of a cell at the next time step differs based on the cell's current state:

- Living Cell:
  - Survives if two or three neighbouring cells are alive.
  - Dies from solitude if one or less neighbouring cells are alive.
  - Dies from overpopulation if more than three neighbouring cells are alive.
- Dead cell:
  - Is born if exactly three neighbouring cells are alive, otherwise remains dead.

2.3 Related Work

2.3.1 Games of Life

There are multiple online versions of the traditional *Life* simulator [20, 21, 22]. There also exists a *Life Wiki* website [23] where enthusiasts collate newly discovered seed configurations, categorise the collection and detail who discovered them and how each configuration evolves over time. The current database on the site is comprised of over fifteen hundred starting configurations.

There are a far more limited number of multiplayer approaches to *Life*. A framework for how a multiplayer game could work is set out on Cornell's Math Explorer's Club website [24], and involves using three cell states - dead, and one for each of the players (denoted by a unique colour for each player). Cells can change ownership if their surrounding neighbours predominantly belong to a different player. If there are an equal number of neighbouring cells owned by other players, the cell becomes neutral but is still alive. Players take it in turns to either kill a living cell or add a new living cell of their colour. The aim of the game is to eliminate all of the living cells of the other players in order to win.

An online version of the Cornell framework exists on the *Life Competes* site [25], however it has not managed to maintain an active user-base. This is most likely in part due to the open-ended game cycle which means that no player ever wins a game. There are also limitations on how many cells a player can place at a given time, and this can make the game feel slow paced likely leading to a reduced level of user engagement. For these reasons, it lacks a motivational drive and competitive edge.

The most successful multiplayer *Life* game is Cary Huang's *Game of Life and Death* [26]. It exists as both a JavaScript web application [27] as well as an application for smartphones and tablets. It follows the Cornell rule-set more strictly than *Life Competes*, is restricted to two-players, and features local multiplayer as well as AI computer opponents of varying difficulty. The key difference between *Game of Life and Death* and *Life Competes* is that the the win condition of the former is to wipe out your opponent's living cells, whilst the aim in the latter is to maximise your own living cell count.

Where *Automata*, the game developed during the course of this project, deviates from other *Life*-related games is in its use of seed configurations as playing cards, furthering the players' familiarity, knowledge, and understanding of the seeds, whilst also being significantly faster paced whilst implementing a Pong-style scoring system to determine the victor. This mixture of intrinsic, educational rewards and extrinsic, competition-driven rewards seeks to provide the motivational drive to ensure longer-term engagement with the game.

The closest game to *Automata* that has been conceptualised is the fictional game *Life*, based on Conway's original conception, that is described in David Brin's 1993 Science Fiction novel *Glory Season* [28]:

> *The paddles sensed the status of neighbouring tiles during a game, so that each piece would "know" whether to show its white or black face at a given time... The game was played as a battle between two teams. Their objective: to lay down starting conditions such that when play commenced, the sweep of shapes would carry their way, wiping clear their opponents' territory, so that the last oases of "life" would be on their side of the board.*

2.3.2 Life Lessons

Far more ubiquitous than variations of the *Game of Life* are educational and resource-rich sites for learning about both *Life* and CA in general. As previously mentioned, the largest collection of information on *Life* can be found on the *Life Wiki* website [23]. As the purpose of this project is to create a novel game, linked with an introductory workshop on CA, it is highly important to recognise sites like this as fantastic resources rather than existing solutions. Indeed, much of the work undertaken in this project would not have been possible without the contributions of hundreds of mathematicians and enthusiasts who have discovered the large range of seed configurations available on the site today.

Whilst many of the available sites have thoughtfully designed user-interfaces, few of the sites offer both interaction with the given *Life* examples coupled with a simplistic explanation that does not seek to delve immediately into the deeper complexities that *Life* can offer [23, 24, 29].

The aim of the educational aspect of this project will be to maintain a sense of simplicity and allow the user to discover the complexities that lie within the systems of the game. In other words, to equip users with the foundational knowledge required to understand the mechanics at work, but not to over-complicate the material presented. The motivation for this is to capture the curiosity of the user through interaction and experience. The intended outcome being akin to how Brian Eno described his own introduction to Conway's *Game of Life* [30]:

> *I was hooked immediately by the thing that has always hooked me — watching complexity arise out of simplicity.*

2.3.3 Strategy Card Games

When deciding the format and play style of the game, inspiration was drawn from several pre-existing and well known games. There are certain degrees of overlap with Tetris, through the visual representations of the *Life* seeds. Although not a card-game, it is worth noting that there is a visual resemblance and to an extent, due to the destruction of blocks of cells, a mechanical resemblance. The main sources of inspiration for the general user-interface of the game were drawn from the card game *Gwent*, which features as a mini-game within the open-world role-playing game *The Witcher 3*. For card layouts, both the *Pokémon* trading card game and the many variations of *Top Trumps* were highly influential. The focus of this section, however, will be on the game loop found in *Gwent*.

Gwent uses a deck of cards, uniquely possessed by each player, with each card representing a different character from the wider game's fictional setting. Cards are split into three categories: close combat; ranged combat; and siege cards. Each card has a point value assigned to it, which is notably visible at the top left hand corner of each card, a title and a brief description that takes the form of a quotation. At the start of a game, ten cards are randomly selected from the player's deck and these make up their hand. The player may choose to discard up to two cards and redraw replacements. On each turn, the player may play a card or *pass*. Once a player has passed, they are unable to play any more cards in that round.

When a card is played, it is placed on one of three rows - each row representing the different aforementioned card categories. Each row has a tally of the total point count of all the cards that have been played. Special cards (*weather cards*) can be used to reduce the point score of all cards in a row. When both players have used all of their cards or have chosen to *pass*, whichever player has the highest total point score across all three rows will win the round. The game is won when a player has won two rounds. It is a fast paced and highly strategic game, and for these reasons acted as a strong guide for how to create a card game with those characteristics.

Figure 2.1: Gwent Strategy Card Game showing a mid-game board state

2.4 Approaches to Educational Game Design

Significant predictors of educational outcomes include motivation and time spent on task [31, 32]. Video games can be powerful tools for increasing time spent on a task, with the extrinsic motivational rewards having the potential to trigger addictive cycles of behaviour [33]. Whilst the current project does not seek to explore the design patterns that drive these addictive behaviours, it is an important factor to take into consideration when making design decisions. This is of particular significance given the research findings of Lepper et al. on how external

reinforcement can degrade intrinsic motivation in users [34]. It is, therefore, the aim of this project to only reward the user through competitive success, as a tool for the user to assess their performance, rather than by involving any further reward system. Malone [8] outlined his theory of how intrinsically motivating instruction can be developed, basing it on *challenge, fantasy* and *curiosity*. These principles will underpin the approach to design in this project.

Malone notes that challenge requires a degree of uncertainty in outcomes. This is an inherent element of *Life*, as one of the criteria that Conway set out was that there should be no initial configuration for which a mathematical proof could exist to show that the population can grow without limitations. Furthermore, the two-player aspect of the game ensures that there is a degree of control that lies outside of the single user's locus of control. This is a factor that is lacking in *Life* but is extremely important in *Automata*. Challenge is a requirement for triggering motivation, as if a task is too easy to complete then it will not create any sense of accomplishment. Piaget [35] notes that this is best achieved through *practice games* where the repetition of an activity enables the user to develop a skill whilst applying it. Csikszentmihalyi [36] stated that activities are required to have clear criteria for users to evaluate their performance and that there should be feedback provided to the user on completion of the activity.

Fantasy is the principle that provides the user with a narrative for which their experience is embedded. It enables the user to feel that what they are participating in is 'play' and not work. For *Automata*, this is produced by the notion of the seed configurations being organic, and being comprised of living cells. The unpredictable patterns of evolution that occur when different cells interact with each other, for example when one seed is consumed by another seed, triggers the user to generate a narrative of one organism eating the other. This anthropomorphic fantasy creation can be further developed through the playing cards' description of the seed configuration. The majority of what will drive the sensation of fantasy for players is borne of the user-interface design, rather than the mechanics of the game.

Curiosity, as mentioned previously, must be nurtured through providing sufficient information to enable understanding of the core mechanics whilst also creating the space for the user to explore and experience the complexity that comes through interactions between different seed configurations. Berlyne [37] outlines his view that it is conceptual conflict that drives curiosity. This means that a belief held is challenged by new information. The unpredictable nature of seed interactions should be a useful tool for driving this element of the motivational design aspect of the game.

One idea that has also played an important role in the development of *Automta* is the notion of *procedural learning* [38]. This is the process whereby the educational outcomes of a game are built-in to the game's mechanics, making learning less instructional. Games that achieve this have been shown to be more effective at motivating and educating users than games that did not. This evidence supports the approach taken for *Automata*, where understanding of seed interactions, and the hidden complexity therein, is developed through repetitive play.

Chapter 3

Development Work

This chapter describes the development stages for creating the *Automata* game and the web application that it is hosted on. It covers the rationale behind the decided-upon technical solutions for both the game and the app, and then gives a detailed breakdown of how the game was designed, the preliminary research undertaken and external tools developed to create the data set required to populate the game with *Life* seed configurations before giving a thorough step-by-step account of the game and application's development cycles.

3.1 Technical Solutions

3.1.1 Game Engine

The primary game engines that are free to access, robust, and dependable are Unreal Engine, using C++, and Unity Engine, using C#. Given the requirement for the game to be hosted online, it would need to be able to built as a WebGL application. It also would need to have as limited a back-end engine as possible so as to remove the requirement on users to have to download and install a larger game application. For these reasons, Unity Engine was the best solution available. It allows developers the choice of what packages to include and, at its most basic, has a minimal build size.

Unity offers users a choice of 2D and 3D starting templates for their projects and, given that the playing-cards and the *Life* board only require a two-dimensional representation, this was the decided upon route. 2D Unity applications also benefit from a smaller build size than their 3D counterparts. Whilst Unity does offer some online multiplayer features, there are significant restrictions on how these work in a WebGL build. This is due to the interplay of security protocols of modern web browsers and the need for at least one user's game instance to act as a server. Details on how these barriers were overcome are detailed in section 3.5.9.2.

3.1.2 Web Application and Hosting

With the aim of having the web application host the entirety of the project, including the game, the workshop and user-evaluation surveys, a React application coded in JavaScript and hosted on AWS Amplify was the best solution. Whilst other solutions were available, the following advantages were deemed to be significant:

- React-Unity-WebGL: A library for hosting the Unity WebGL build inside a React application, enabling interactions between the web app and the game instance.
- React Router: A library that enables multi-page React applications, where browser navigation buttons do not cause the user to lose track of where they are in the application. This felt of particular importance for the workshop section of the site, especially if secondary school educators chose to run the workshop with students who may accidentally navigate away from the page and this enables them to easily return to where they were.
- SurveyJS: A library for creating customised surveys, allowing the developer to have control over visual representation through a CSS style sheet. The output of the survey is a JavaScript Notation Object (JSON) which can be easily used in NoSQL databases.
- GraphQL and Database integration: As the intention was to include user-evaluation within the site, the ability to create back-ends for React applications, using API keys to access and write to DynamoDB tables through the use of a GraphQL schema, offered simplicity and ease of application.
- Functional components: Learning pages on the site would be easy to template and then customise, reducing duplication of work carried out.
- AWS Amplify works well with React applications and gives the developer access to AppSync functionality. The *Free Tier* gives sufficient allocations of resources to prototype, test, and deploy the application without incurring costs. It enables access to databases and keeps API keys protected in the process.

The decision to use AWS-hosted React was one that was taken with not just the immediate benefits in mind, but also to allow for the longer-term potential of scaling up both the educational workshop and the game in any future work.

3.2 Game Design

The conception of *Automata* was based on the idea that *Life* could be reworked into a far more competitive and engaging game. The motivation behind this was driven by the research findings previously set out on how challenge is seen as key to producing positive outcomes for learners [8, 35]. In order to achieve this sense of competition, there needed to be a way in which points could be scored against the opponent. Given the most common forms of *Life* already utilise a Chess style square-grid based board, this was a natural starting point for the game board. Scoring, then, could take the form of: domination through cell count, an idea which had already been attempted with varying degrees of success in other games; or the more traditional format of having a scoring zone, such as in games like Pong. The scoring zone approach was best suited to a card-based game as it opened up the game to the potential of cards being either offensive (seeking to score points) or defensive (seeking to prevent points from being scored).

Each player has a deck of playing cards, with each card representing a different seed configuration. The cells that comprise the seeds are the tokens that are used to play the game. A player move involves selecting a card from the player's deck, previewing the placement of the seed configuration on the game board and then confirming this placement. During the initial stages of conception, the number of moves allowed by each player on each turn was deemed to be a decision that would have to be taken after a degree of preliminary play-testing. An important principle of CA is the notion of uniformity - that changes happen in parallel. This principle dictated the requirement for moves to occur between generations. In practice, this meant that the move-based decision-making of the player would happen when the board was in a resting state.

One of the key elements that is lacking in other *Life* games is the ability to respond and adapt to the opponent's moves. With the introduction of moves that are either offensive or defensive, it logically followed that there were two types of adaptation that could occur during a game: responding to an incoming attack in order to neutralise it; or exploiting a lack of defense with an offensive card. A decision was made to have both players carry out their moves in unison, to allow risk-taking to take on a more prevalent role in the game.

Games such as Chess are *Perfect Information* games, because when a player makes a move they do so with complete knowledge of the state of the board and the potential actions that their opponent can take. Games such as Poker do not present this knowledge to the player and therefore are known as *Imperfect Information* games. Both Chess and Poker are turn-based in their approach to player moves, and this is where *Automata* deviates from them. By having players make moves at the same time, both players have an equal understanding of the state of the game, but they both lack the knowledge of the state of the board immediately following their move. This is called a *Simultaneous Game*, and is distinct from both *Sequential* and *Continuous* games. To ensure that this did not have a detrimental impact on the game, and to allow for a degree of prediction of the opponent, the decision was made to have deck cards face up. Both players would, therefore, know what possible moves their opponent could make, and therefore the game would involve both anticipation of moves (proactive adaptation) and reactions to previous moves (responsive adaptation). This follows other examples of *Sequential* games, such as the Prisoner's Dilemma or Rock-Paper-Scissors, where each player knows the choices that their opponent can make, but is left to make assumptions as to what choice they will make. Consideration was also given to the work required to create an AI opponent to play against, and by having cards face-up it would make the task of simulating the players' potential moves significantly more achievable.

These decisions enabled a basic concept to be defined for *Automta*: a simultaneous, face-up card-based game where each player seeks to score points by having game tokens cross their opponent's edge of the game board in order to win.

3.3 User Interface Prototypes

3.3.1 The Game View

Once the outline of the game's mechanics had been decided upon, the next step in the development process was to draft the game view. The key information that the player needed to have access to included the current state of the game board and the cards available in both players' decks. A mock up of the view was created in MS Paint [Fig. 3.1].

3.3.2 Playing Card

The next game object that was drafted was the playing cards. The presented information requirements for the cards included:

- Name of the seed configuration
- A preview of the seed configuration
- An information panel with gameplay-related details
- Buttons to interact with the card

Inspiration for how to present this information came from other strategy card games such as *Pokémon* and the previously mentioned Witcher-based game *Gwent*. In addition to the requirements listed above, it felt that it was important for the seed configuration preview to not simply be a static image but a *Life* instance itself, allowing the player to simulate the seed in order to have an understanding of how the card might interact with the board prior to making a move. The initial mock up was also created in MS Paint [Fig. 3.2].

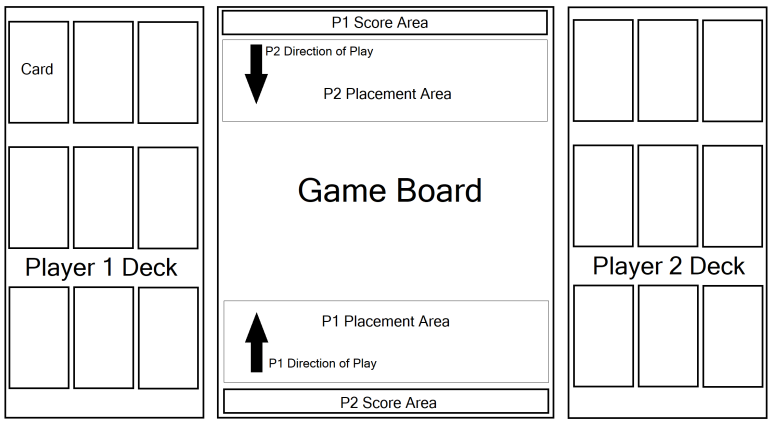

Figure 3.1: Game View Prototype

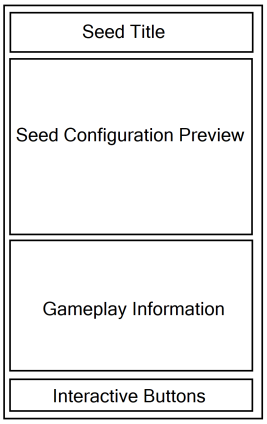

Figure 3.2: Card Prototype

Game view prototype (L) shows the layout of the game board and player decks, with the direction of attack for each player indicated with arrows and the respective score areas shown at each end of the board. The playing card design (R) shows the distinct panels of the card including a preview that can simulate the seed configuration and buttons for expanding either of the two panels to fill the card.

3.4 The Universal Seed Bank

With the initial sketches of how the in-game objects would be visually represented, and before any development work on the game itself began, the game needed to have a bank of seed configurations that could be used to populate the playing cards. These were sourced from pre-existing online archives, however they needed to be converted into a universal format that could be used within a query-able database, and therefore a parser was created for this purpose.

3.4.1 Java Parser

The *Life Wiki* site [23] was an extremely valuable resource for this work. The site catalogues over fifteen-hundred configurations, and has a downloadable archive of over five-thousand Run Length Encoded (RLE) pattern files. The RLE files are human-readable representations of the seed configurations, that contain the name of the seed, any comments on who discovered the seed and a short description of the seed (including how commonplace the configuration is, for example), the dimensions of the grid that contains the starting configuration, the *Life* rule-set that the seed is used for and a string that encodes the seed configuration. The string is encoded using alternating state tags (alive or dead) preceded by a run count (how many cells of this state follow on from one another moving from left to right along a row. When the encoding reaches the end of a row, a dollar sign symbol is used to represent the end of the row. If fewer cells were denoted than the seed's width, it follows that all additional cells in the row are dead.

```
#N Gosper glider gun
#C This was the first gun discovered, as its name suggests, by Bill Gosper.
x = 36, y = 9, rule = B3/S23
24bo$22bobo$12b2o6b2o12b2o$11bo3bo4b2o12b2o$2o8bo5bo3b2o$2o8bo3bob2o4b
obo$10bo5bo7bo$11bo3bo$12b2o!
```

Listing 3.1: RLE Example

These RLE patterns represented the majority of seed files, however some patterns were encoding as plain text files. They feature the same information, however the grids were represented using a simple system of full-stop symbols for dead cells and a capital O for living cells. The same rule applied whereby the remaining cells to the right of the last living cell in a row were assumed to be there and are not required to be listed. This rule is not always utilised, as shown below.

```
!Name: Gosper glider gun
........................O...........
......................O.O...........
............OO......OO............OO
...........O...O....OO............OO
OO........O.....O...OO..............
OO........O...O.OO....O.O...........
..........O.....O.......O...........
...........O...O....................
............OO......................
```

Listing 3.2: PlainText Example

Whilst this archive represented a hugely valuable data set, what was required for this project was the creation of a universal format for storing seed configuration information. It needed to be expandable, based on any additional information that could be included on the seeds from automated or manual analysis, and given the vast number of seeds available (far beyond the numbers required for a card game) it needed to be readable by a database system so that the total number of seeds could be refined through querying. For these reasons, the JavaScript Object Notation (JSON) format was decided to be the optimal format.

In order to convert the RLE and PlainText files to JSON, a parser and writer were created using Java and coded in IntelliJ IDEA. The choice to use Java was based around the functionality that the language has for working with JSON objects, and the ability to build and run the program locally. Full details of the program can be found in the project's repository. Each file was checked for a file ending and then was passed to an appropriate conversion object. The conversion object would then work through each line of the file, checking for the initial characters to determine what the line would contain (e.g. meta-information in RLE files is always preceded with a hashtag followed by a category identifier such as N for name, or C for comment).

Once meta data had been stored inside a newly created JSON object, the grid configuration was converted to a Boolean array list. The array list was chosen as the datatype for this as it is a flexible collection type to use when adding new data as each new line is read in. An additional meta-information data point was created during the conversion, where the number of living cells in the seed configuration were counted. This was included in the resulting JSON objects as a means of improving the initial filtering of the data set.

Once the Boolean values had been extracted from the files, the array was encoded as a Base-64 string [List. 3.3]. This was decided upon as it is a relatively compact way of storing information, with each character in the string representing six Boolean values and there being widespread compatibility and in-built encoding and decoding functionality in many programming languages. Using this format as a universal encoding of the seeds and game boards would also facilitate sending game state-related messages between game instances during the development of online gameplay. The array list was therefore converted to a Byte Array before being encoded as a Base-64 string to be added to the JSON object. This object was then added to a JSON array, representing the seed files that had been converted, and once all files had been converted, the array was written to a new JSON file using Java's stringify functionality.

In the process, any files that were duplicates (i.e. the same seed was encoded in both RLE and PlainText) were skipped past. Additionally, any seed grids that were of a scale that would either cause memory issues when building the Boolean array list, or were simply too large for the present purposes, were discarded. This resulted in a data set of three-thousand two hundred unique seed configurations. The resulting JSON file was used to create a MongoDB database and the refinement process began.

```
public String convertBoolToString(ArrayList<Boolean> bList) {
    int arrSize = bList.size();
    int byteArrSize = calcByteArraySize(arrSize);
    byte[] byteArr = new byte[byteArrSize];
    for (int byteNum = 0; byteNum < byteArrSize; byteNum++) {
        for (int bitNum = 0; bitNum < 8; bitNum++) {
            if (byteNum * 8 + bitNum >= arrSize) { continue; }
            if (bList.get(byteNum * 8 + bitNum)) {
                byteArr[byteNum] |= (128 >> bitNum);
            }
        }
    }
    return Base64.getEncoder().encodeToString(byteArr);
}
```

Listing 3.3: Base-64 Encoding

3.4.2 Sifting Seeds

The number of seed cards that could be used for the game was determined, based on previous and popular card games such as a standard deck of playing cards (52) or *Pokémon* (initially 151), to be greater than fifty but less than two hundred. This felt like a manageable number of cards, where players would be likely to encounter the same cards on a regular basis when repeatedly playing the game. The key requirements for querying the seed configurations were based on the in-game playing cards. The seed grid would need to fit, in its entirity, on to the preview panel. This would create strict limitations on the dimensions of the grid. Furthermore, all seeds needed to follow the same standard rule-set for *Life* (B3/S23, as outlined in section 2.2). Queries on the starting data set in MongoDB elicited the following results.

Filters				Seed Count
Living Cells	Max. Height	Max. Width	Rule	
0 to 20	NA	NA	NA	836
0 to 20	NA	NA	B3/S23	781
5 to 15	10	10	B3/S23	577
5 to 15	6	6	B3/S23	237

Table 3.1: MongoDB Query Results

The final figure of two-hundred and thirty seven seed configurations was a suitable number to work with. All of the starting grids would fit onto a small card-based preview, and there were sufficient numbers to further refine and filter out cards that either did not work well in the game or for which there were too many that filled a similar purpose. The next stage of development was to begin building the *Life* simulator in Unity, in order to run an automated analysis of the seed configurations so as to build up a greater bank of meta-information. It was decided that this process should initially be automated, rather than run manually, with any further refinements or adjustments being made on a per-seed basis.

3.5 Core Development Cycle for Automata

This section details the sequence of development work that was carried out to create *Automata* in Unity. The functionality was roughly introduced in the order presented here, whilst iterations and improvements of functional components, based on informal user evaluation, were often rolled out in parallel. The details of any iterations are given within the section that describes the development of the component to which they relate. The overall process required the creation of a *Life* simulator, which was subsequently used to drive the automated analysis of the collated seeds for further refinement, followed by the design of the game's user interface and menu. A description is given of the overarching hierarchy of the Unity project before an in-depth breakdown is given of the core components that implement the game's mechanics and different game modes.

3.5.1 Simulating Life

The first stage of development within Unity was to create the functionality to simulate the *Game of Life* using the rule-set described in section 2.2. A decision had to be taken on how to represent the board in memory, and consideration had to be given to the impact that this decision would have on future functionality. Options included using:

- A 2D Boolean array, which would have notable benefits for code readability but uses greater amounts of memory than other data types.
- A 2D byte array, which would be more memory-performant than a boolean array due to bit-packing but would also require bit manipulation in order to access individual bits. This approach would also potentially lead to excessive memory allocation due to the fixed number of bits each data point can hold (e.g. 60 cells needed, requiring 7 bytes and leaving a remaining 4 bits unused). The same benefits and limitations apply to other bit-packing-based approaches such as using character or integer arrays.
- A 1D bit array, which is memory-performant with each bit using only one bit in memory and has the added advantage of having a significant degree of in-built functionality in C# for manipulating the data. This functionality includes the ability to run AND, OR and XOR functions on the bit array, as well as making the process of reversing the bit array highly simplistic.

Given the in-built functionality, which would prove to be extremely valuable later on during development when creating multiplayer functionality, the 1D bit array approach was decided upon. Whilst it does require a degree of tracking of rows and columns of the game board grid, clear labelling of variables would reduce the impact that this has on the readability of the code base.

The next decision required was how *Automata* would approach the concept of the game board wrapping. This means continuity from one edge of the board to another, whereby the left most cells left-hand neighbour are the cells on the right side of the board. Given the limited grid size available and the diagonal movement trajectory of some of the most common seed configurations (e.g. gliders), it made sense for the board to wrap left-to-right. No wrapping is used for top-to-bottom as this is the scoring zone area and it would not fit with the wider concept of the game having a Pong-style goal to pass through and leave the board in the process of scoring points. The core methods that drive the life simulation are given below.

```
private int MooreCount(BitArray brd, int row, int col) {
    var neighbours = 0;
    for (var i = -1; i <= 1; i++) {
        for (var j = -1; j <= 1; j++) {
            var isInvalidNeighbour = (
                i == 0 && j == 0 || i + row < 0 || i + row >= fullRows
            );
            if (isInvalidNeighbour) { continue; }
            var cellIndex = GetCellIndex(i + row, (j + col + bCols) % bCols);
            neighbours += brd[cellIndex] ? 1 : 0;
        }
    }
    return neighbours;
}
```

Listing 3.4: Moore Neighbourhood Count

```
private void NextGeneration() {
    var nextBoard = CreateBoard();
    for (var i = 0; i < board.Length; i++) {
        var count = MooreCount(board, i / bCols, i % bCols);
        nextBoard[i] = !board[i] && count is 3 || board[i] && count is 2 or 3;
    }
    board = nextBoard;
}
```

Listing 3.5: Creating the next Generation

These functions run within a timer-driven loop, triggered by the *Update* method in Unity which is called for every rendered frame. The timer makes use of Delta Time so that it is not affected by variations in users' frames per second. The next task was to represent the board visually. For this, a TileMap object was used, offering not only in-built access to coordinates but also enabling mouse hover-over functionality. A separate class was created to handle visual representations, in order to not overburden the life simulator class in anticipation of the significant amount of additional functionality that it would need to contain. This also enabled UI design choice presets to be easily accessed and stored in a location distinct from the variables that are used for manipulating the game's mechanics.

3.5.2 Seed Analysis

The next stage of the project was to run additional automated analyses on the two hundred and thirty seven seed configurations. This required being able to load a seed from a JSON file and access the Base-64 string that encoded the starting configuration as a bit array. To facilitate the loading and storing of information to and from JSON files, a serializable class was used for the *Seed* class. This enables every public variable inside the class to be automatically written to and read from when calling methods from the in-built JSON Utility class. In addition to this simplification, and given the regular requirement for the creation, comparison, and manipulation of bit arrays across the project, a static class was created to handle bit array specific functionality in order to reduce code duplication.

The process of seeding a game board with an initial seed configuration included: conversion of Base-64 string to a *seed bit array* (of the same dimensions as the seed's height and width); validating the resulting array; creating a new empty bit array of the same dimensions as the game board; translating the *seed bit array* to the blank *seed board bit array*, based on row and column coordinates (passed as method arguments); and then finally running an OR comparison on the current *game board bit array* and the newly created, equally sized, *seed board bit array* to create the resulting game board with the seed overlaid.

Whilst essential for the analysis, this functionality laid the foundations for future development allowing players to place their seed configurations on the game board. The automated analysis of the game board is triggered by activating a Boolean value which is exposed to the main editor, allowing the user to turn it on or off as required. The output of the analysis was a newly written JSON file which maintained all of the information previously collated on the seed configurations, whilst also adding in new meta-information. These new variables were the lifespan of the seed configuration, the potential player score achieved through cells crossing the up-most border of the board, the potential opponent score achieved through cells crossing the bottom-most border of the board (in case some configurations required rotation), the minimum and maximum living cell counts, whether the seed becomes a stable still life, whether the seed modulates by repeating the same configuration over a given number of generations, and then based on these additional variables, whether the seed can be determined to be a potential offensive card or defensive card.

The resulting JSON file was used to create a new MongoDB database, and when queried to filter only offensive seeds, there was a resulting twenty-five configurations that could potentially be used as *Attack* cards. This number, after some manual analysis of the resulting cards, did increase slightly. There were two-hundred and three defensive seeds, and ten seeds that did not fit the requirements for either defensive or offensive categories. These ten seeds are generally chaotic in their evolution and have been included as a separate *Mixed* category, often showing to be extremely disruptive to the current board state. To balance the deck, cards were either duplicated or discarded to ensure a workable ratio. Ninety three cards comprise the final deck.

3.5.3 Gameplay User Interface Design

Before the project could progress further, time was allocated to progressing the development of the user-interface and visual elements of the game. Unity allows for the creation of *prefabs* which are a group of game objects belonging to a single parent object. They can be instantiated at run time and each instance can have its exposed variables manipulated to create variants. A *prefab* can also contain within it nested *prefabs*. This was ideal for designing and iterating on the user-interface, as the hierarchical structure enabled object-level changes that had been made to the card *prefab* to affect all cards within the player deck *prefab* whilst maintaining the ability for each card to hold unique values. This approach was used across the project, resulting in a cascade of nested *prefabs* that could, non-destructively, be iterated upon [Fig. 3.3].

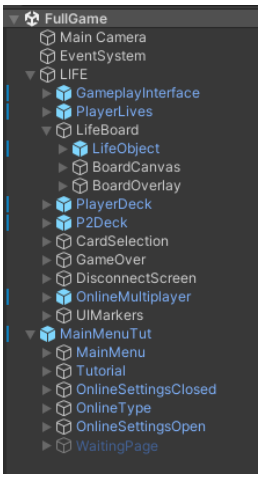

Figure 3.3: Project Hierarchy
Blue colouration indicates prefabs

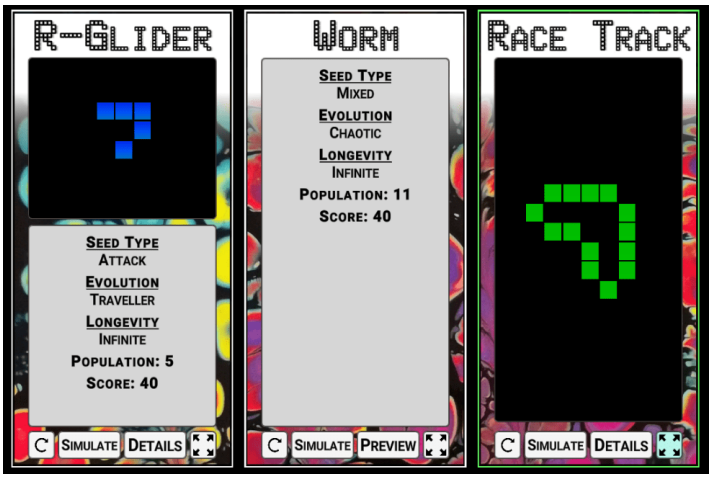

Figure 3.4: Card Designs
Initial state (L); expanded details; selected card and expanded Life preview (R)

The playing card design was based around the original concept (see Fig. 3.2), being comprised of four distinct elements: title; *Life* preview; meta-information; and interactive buttons. The preview element uses the same functionality, both visually and mechanically, as the game board. The buttons needed to be able to trigger the preview to start simulating, to reset the simulation back to its original state, to expand the preview to give the user a better view of the preview grid and to expand the details panel. The first iteration of the design was shared with three individuals, one of whom has colour blindness, for informal feedback on layout, colour palette and readability. The initial colour choices, a pale pink colour used for the background of the details panel, was discarded as it was determined to be difficult to view for people with colour blindness. The font choices and alignment of text were also refined based on this informal feedback, with the resulting iteration being confirmed, by those previously asked, as being more readable and easier to view. The final version of the *Seed Cards* included a randomised background based on the type of card. Attack cards have blue-green backgrounds, whilst Defensive and Mixed cards use a red-black background. The backgrounds are sliced sections of a free-to-use abstract painting [39].

The colour of the cells in the *Life* preview needed to be bright and easily distinguishable against a black background. The card's outline and preview cells also needed to change colour when the card was selected to ensure that users had a clear visual identifier of which card they were currently previewing from their deck. This can be seen in the right-hand most card in Fig. 3.4. This meant that a total of four colours were needed for the selected and unselected card states for each player. The final choice was to use blue (unselected) and green (selected) for player one, whilst player two would have red (unselected) and yellow (selected). The unselected colours, both being primary colours, are the colour choices for cells on the actual game board. In order to give the cells a more animated and life-like visual element, a vertical-gradient interpolation shader was created that creates a waterfall-like ripple of brightness on each cell in unison. This effect was also applied to the selected card's border.

The next stage in designing the user-interface was to create the game view. This is the scene that player's see whilst they are playing a game of *Automta*. Based around the initial sketch (Fig. 3.1), there was an important mechanical decision to be made that would strongly influence how the game view would be designed - the size and cell-based dimensions of the game board. The first iteration of the board (Fig. 3.5) used a fifty column by fifty-four row grid (this included two scoring zone rows for each player). This enabled both players' decks to fit either side of the game board on one screen. After some informal user-testing, where games would often result in a tie-game, it was determined that the square game board made it notably more difficult for players to be able to score points against their opponent. As a result, the game board was widened to be eighty cells wide and fifty-four cells long. However, this had an impact on the game view as both player decks would no longer fit into one scene whilst maintaining readability of the playing cards. As a result of this process, the decision was made that players should use the *Tab* key to shift the play view in order to be able to view the opponent's card deck. This meant that an optimal solution could be found in terms of both the game's mechanics and the user-interface's readability. Examples of the original and final state of the game view are given in Figures 3.5 and 3.6 respectively.

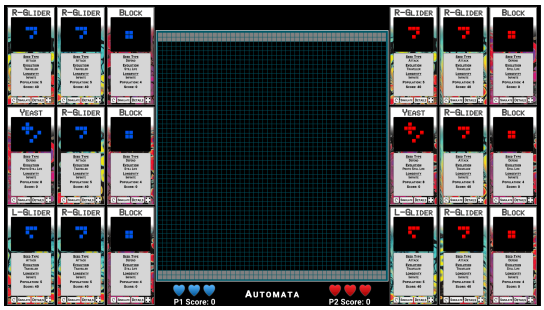

Figure 3.5: Original Game View

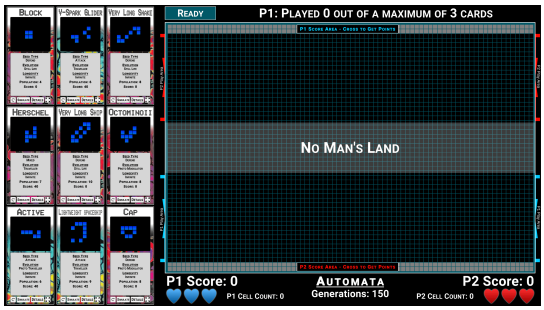

Figure 3.6: Updated Game View

The first iteration of the game view was deemed to be overcrowded and the restricted board size often led to stalemate games. The wider, updated version of the board and game view had benefits to both the game's mechanics and space for providing game-state information.

3.5.4 Menu Design and Scene Management

The sole requirement for the menu was that it needed to be extremely easy to navigate. The only functionality required was to be able to choose between different game modes and the *How to Play* tutorial. On loading the game, the menu is the first screen that is seen by the player, and therefore the only other required element was the game's title. The initial concept during early stages of development had a single game mode option (local multiplayer game) and the tutorial. This was expanded to include additional game modes as they were developed. The final design for the game menu is shown in Fig. 3.7.

Figure 3.7: Game Menu

Font selection has been kept consistent throughout the game. Titles of menus and cards all use the same cell-tile styled font as shown above. All other text elements use a font that was, during informal user-evaluation, determined to be the easiest font to read. The use of *small capitals* for non-capitalised letters improved the letter spacing and, particularly on devices with smaller screen sizes, improved the readability of text on playing cards. Colour palettes are identical across buttons on both playing cards and menu items. Only the game view's *Ready* button has a unique background colour so as to draw attention to it (see Fig. 3.6).

As development progressed, developer-driven testing on a range of different devices flagged performance concerns relating to the game's scene management. Whilst the original approach had been to have the menu, the tutorial, and each game mode contained within separate Unity scenes, this drastically increased the size of the resulting WebGL build. Due to memory limitations, particularly on browsers such as Google Chrome, this had the unintended consequence of preventing the Unity game from loading on some older laptops. A complete re-working of the project was therefore required, with the entire game now being contained within a single scene. This re-use of game objects significantly reduced the size of the game's build and has enabled it to load successfully on older devices and to load faster across all devices. This required a notable degree of debugging work to be carried out as, in the event of a player quitting a game, the entire scene needed to be reset to a default state. This is something that had previously been automatically achieved when using a multi-scene approach. The scene management view of the full game can be found in the Appendix [A.1].

3.5.5 Game Object Hierarchy

The hierarchy used for the main elements that comprise the game view was extremely important in order to reduce unnecessary duplication of objects in the scene. Due to the fact that the entire game exists within a single scene, it also facilitated the process of moving UI elements as the player navigates through different menu's and game screens. Any scripts that required access to multiple objects needed to be at a top-level in the hierarchy, and likewise any objects that needed to be accessed by multiple objects needed to be within a direct upward traversal through parents to reduce inefficiencies in accessing components. The basic hierarchy is as follows (previously shown in Fig. 3.3):

- ➔ Gameplay Parent Object: Game Manager and Seed Reader
  - ➢ Gameplay interface: Gameplay data and interaction
  - ➢ Game Board: TileMap, Life simulator and visualizer
  - ➢ Game Board Overlay: Additional UI overlays
  - ➢ Player One Deck: Deck manager
    - → 9 playing cards, each with their own Card Manager component
  - ➢ Player Two Deck: As per Player One Deck
  - ➢ Card Selection Screen: Unique variant of a player deck *prefab*
  - ➢ Game Over Screen
  - ➢ Online Game Disconnect Screen
  - ➢ Online Multiplayer Components
- ➔ Menu and Tutorial Parent Object: Menu Functionality
  - ➢ Main Menu
  - ➢ Tutorial
  - ➢ Online Multiplayer Settings

The *Game Manager* is a significantly important script which handles the main game loop, holds game state variables and manages the position of child UI elements. It regularly needs to access components in child objects as well as be sent information from other scripts. Its position at the top of the hierarchy enables this to work seamlessly. Similarly, the *Seed Reader* is the script that is responsible for loading the seed configurations and meta-data from a JSON file. By having a single instance of this at the top of the hierarchy, all card instances can access seeds through an index number so as to reduce duplication of this data across the scene.

Deck Managers are responsible for handling the interactions between the playing cards and the *Game Manager*. To reduce code duplication, the same *prefabs* are used for both players with each instantiated deck having unique variable presets to distinguish between the players. When starting a new game, a unique variant of the *Deck Manager* is used on the card selection screen. This sends information on the selected cards to the *Game Manager* which then disseminates this information to the appropriate player *Deck Manager*.

3.5.6 The Card Manager

A player's deck is comprised of 9 playing cards. Each playing card represents a seed, with its associated meta-data. Playing cards have four main states: a state of belonging to player one or belonging to player two; a state of either having been played (disabled) or still being able to be played (enabled); a state of being selected (triggering alternate colours for the *Life* preview and card border) or not being selected; being either an Attack, Mixed, or Defend card, based on the meta-data associated with the seed, causing the card's background image to be randomly allocated from a set of spliced background images (grouped into two lists, based on the predominant colour palette of the image slice). The *Card Manager* is responsible for this state management and setting the text elements of the card panel with the seed's meta-data. When a card has been played, a black background and an overlaid stamp ("PLAYED") denotes that it is no longer available.

To maintain separation between game state information and UI interaction, additional scripts were used for managing the events triggered when players clicked on the different card panel buttons and to store the current interaction-state of the cards. For example, the transformations required for expanding or minimising UI elements on the cards, such as the *Life* preview, were kept distinct from the *Card Manager* script so as to facilitate any gameplay debugging that was required.

3.5.7 The Deck Manager

The *Deck Manager* is responsible for holding data associated with each of the 9 playing cards that comprise each deck, and for maintaining mutually exclusive states across the cards. For example, when a new card is selected the card's *Card Manager* notifies the *Deck Manager* that it has been selected and this in turn causes the *Deck Manager* to ensure that the *Card Managers*' of all other cards in the deck are de-selected and return to their default state. During the core gameplay loop, the *Deck Manager* also holds a Boolean list of whether cards have been played or not. This makes the process of ending a round significantly faster due to the in-built functionality of lists (List. 3.6). Additionally, the collation of card data enables the game to ensure: that no duplicate cards exist within a deck; that all cards can be easily reset at the end of a round; and that, at a later stage in development, this facilitated the process of randomly selecting both the card and number of cards for the computer AI opponent to play (List. 3.7).

```
1 public bool AllCardsPlayed() {
2     return cardStates.TrueForAll(b => b == false);
3 }
```

Listing 3.6: Checking if all cards in a deck have been played

```
1 public int GetNumberOfCardsLeft() {
2     return cardStates.FindAll(c => c).Count;
3 }
```

Listing 3.7: Checking remaining cards left to be played

3.5.8 The Game Loop and the Game Manager

The *Game Manager* is the foundation on which the game loop runs. It is responsible for:

- Initiating a new game.
- Setting the game type: local multiplayer, online multiplayer, or single player.
- Updating UI elements with game state data.
- Hiding or showing UI elements based on the current stage of the game loop.
- Storing data associated with player ownership of cells on the game board.
- Triggering messaging between online multiplayer clients and responding to incoming messages.
- Determining when a round is over and when the game is over.
- Resetting the entire game's state back to a default starting point.
- Handling any player input commands and ensuring that they only function when they are allowed to (i.e. cannot shift player view when in a menu).

Whilst this is a significant amount of functionality to contain within a single script, in doing so it significantly reduces the amount of duplicated storage of variables. The code is separated by multi-line comments into distinct sections, covering each of the roles that the *Game Manager* is responsible for. The primary game loop is illustrated in Fig. 3.8 below.

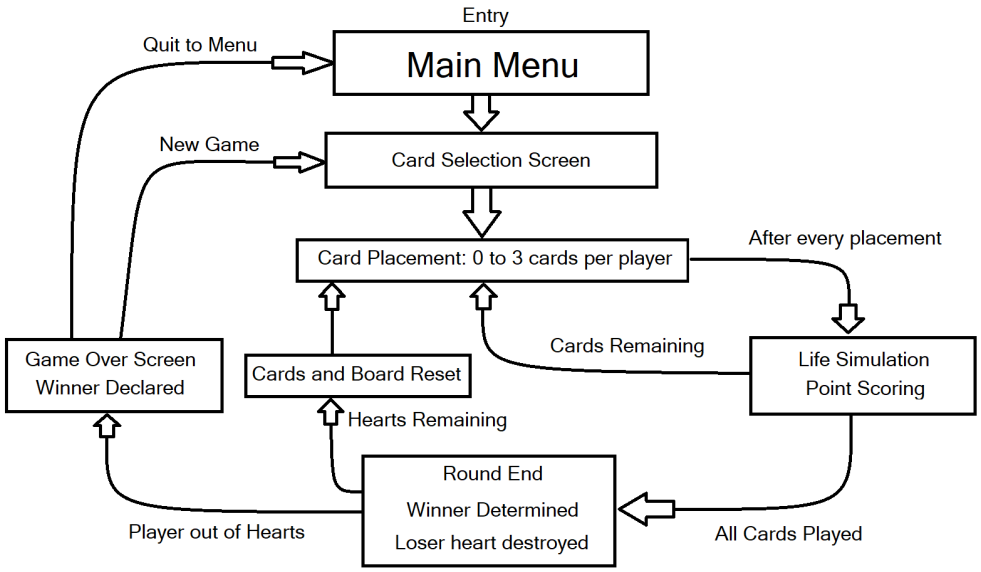

Figure 3.8: Core Game Loop

3.5.8.1 Game Loop: Game Initiation

Games are initiated through the main menu. Depending on the player's choice, a method in the *Game Manager* is called that will set the UI elements to their default state, will set the *Card Selector* to be prepared for either one or two players to choose cards, and then will move the player's view to either additional online settings or to the card selection screen. Full details of online multiplayer functionality are described later in section 3.5.9.2.

3.5.8.2 Game Loop: Card Selection

Using a unique variant of the *Deck Manager prefab*, the card selection screen allows for all cards to be shuffled or for each individual card to be replaced with a randomly selected card from one of the three card categories: Attack; Defend; or Mixed. When cards are replaced, the seed index for all other cards is checked to ensure that no duplicates exist. In a single player game, the computer opponent's cards are randomly selected. In online games, a message is sent to the opponent including an under-score separated list of integers denoting each of the seed indices of the player's chosen hand. Consideration was given, following informal user feedback, of introducing a timer to prevent players from taking too long to choose their cards and introducing a restricted element of randomness in the selections. This has not been introduced in the project to date, with the aim of allowing new players time to read and interact with cards prior to beginning a game. In online games, the player's view will only progress to the game view (of the game board and respective player decks) once both players have confirmed their choices. The card selection *Deck Manager* calls a method in the *Game Manager* passing seed indices and player number as arguments. The appropriate player deck is then updated with the correct seed cards before the game view becomes visible to the players.

3.5.8.3 Game Loop: Making a Move - Card Placement

During development, card placement and previewing was challenging to conceptualise and execute, requiring a degree of consideration to be given to user interaction and immediate visual feedback. Before any card placements are carried out, the current state of the game board may be stored in a variable. How this variable is utilised varies depending on the current game mode. In local multiplayer games, once the first player has confirmed their card placements, the board is reset to its initial state in order for the second player to take their turn *blind* (i.e. without knowledge of what their opponent has done on their turn). In single player games, the variable is used as the starting point for the computer opponent's moves. For online multiplayer, this variable is not required and therefore is not stored.

Card selection and placement required communication between several key scripts: the selected card's *Card Manager*; the card's associated *Deck Manager*; the *Game Manager*; the main game board's *Life Simulator* and *Life Visualizer*. When a card is selected, the card's state is updated to reflect that it has been selected, any other cards are deselected and the visual elements are updated to reflect this. The index of the card's seed in the seed bank is sent to the *Game Manager* and the associated *Seed* is instantiated as a local variable. This is possible as only one seed can be selected at any given time. A Boolean variable, denoting that the player is previewing a seed, is set to true and this value is propagated to the *Life Simulator* and *Life Visualizer* thereby triggering the *Preview Mode*. This functionality cannot be initiated during the generation stage of the game loop or during the end of round winner declaration screen. Players are also prevented from being able to select cards belonging to their opponent, thereby preventing *Preview Mode* from being triggered with an opponent's card.

During *Preview Mode*, the *Visualizer* script's *Update* method, called with every frame rendered, stores the current coordinates of the player's mouse on the game board. The *Simulator* fetches these coordinates and checks if the corresponding grid cell lies within the range of cells that constitute the game board. If the coordinates are within bounds, the *Simulator* stores the existing board state in a temporary variable, converts the selected seed's Base-64 string encoding of the seed configuration to a game-board sized bit array and uses an OR comparison to create an updated version of the game board with a preview of the selected seed at the given mouse coordinates. This is then passed to the *Visualizer* so that the player can view their previewed seed placement. Whilst *Preview Mode* is active, if the left mouse button is clicked then the temporary game board variable is set to null and the *Preview Mode* is deactivated by setting the associated Booleans to false. If a player de-selects their card before confirming placement with the left mouse button, or if their cursor moves off the board or to a new cell, the current game board is reverted to its original state from the temporary value.

Determining and storing the ownership of cells by each player required two additional bit arrays to be used to encode this information, one for each player's set of owned cells. When a player places a card, the game board sized bit array that represents the seed overlay and the player's respective ownership bit array undergo an OR comparison to update their set of owned cells. Cell ownership cannot be overridden by placing a new seed card on the board. This is achieved by first running an XOR comparison with the opponent's owned cells, followed by an AND comparison to keep only the resulting cells that belong to the new seed that is being placed, and then finally running the aforementioned OR comparison. The associated code for previewing cell ownership is given below in Listing 3.8.

```
1  private void PreviewCellOwners(Seed seed, int player, int row, int col) {
2      var seedAsBoard = GetSeedAsBoard(seed, row, col, player == 2);
3      var seedCopy = CloneBoard(seedAsBoard);
4      switch (player) {
5          case 1:
6              player1Board.Or(seedAsBoard.And(player2Board).Xor(seedCopy));
7              break;
8          case 2:
9              player2Board.Or(seedAsBoard.And(player1Board).Xor(seedCopy));
10             break;
11     }
12 }
```

Listing 3.8: Preview Mode: Cell Ownership

As players are restricted to only placing seeds on their side of the game board, it is not possible for seed placement to cause complications relating to placement at the same location. The game board bit arrays from each player's turn are combined using an OR comparison. The *Visualizer* is updated with the end of turn board state and the generation phase can commence.

3.5.8.4 Game Loop: Generating Life

The game board simulates a set number of generations, based on the code in Listings 3.4 and 3.5. The number of generations to simulate was initially set to fifty, and after preliminary informal user play testing this was increased to one hundred generations. After the dimensions of the game board were increased to be eighty-cells wide, this was further increased to one hundred and fifty. This number enabled player's basic Attack cards, such as gliders, to be able to traverse the board to score points within a single turn. Higher scoring and more complex cards, however, are unlikely to score points and therefore the opponent is able to react to incoming attacks that have the potential to score significantly more points that a basic glider.

Cell ownership is determined for each cell, on each generation, by counting the number of surrounding cells owned by each player. Whoever has the greatest number of neighbouring cells will own the living cell on the next generation. If an equal number of owned cells surround a cell, the cell will become neutral. This is visually signified using white cells. Neutral cells cannot score points for either player, but do play an active role in the creation of future generations. A neutral cell is encoded by the value of the associated bit in each player's cell ownership bit array being set to true.

Modern computers are able to simulate generations at an extremely fast pace, and therefore a timer was implemented to slow the progression through the generations. A time delay of 0.07s was decided upon, as this facilitated visual tracking of seeds whilst also ensuring that the generation phase did not last too long (10.5 seconds total). A generation countdown was added, based on informal user feedback, to enable players to know the game's progress through the generation phase.

Scoring occurs on each generation, with the number of living cells belonging to the opponent that exist within the last two rows of each player's side of the board being counted and added to the opponent's score. Visual score trackers (see Fig. 3.6) are updated for every generation. A glider that passes through the opponent's scoring zone, unimpeded, will net the player a total of forty points. However, the best way to score points is for a player to create a still life formation within the opponent's scoring zone. This will score one point for every static cell that sits inside the scoring rows for every single generation. This can lead to significant gains within a single turn, but can then be easily disrupted at the start of the next turn due to the delicate balance that must be maintained for a still life to persist. In addition to the score tracker, counters for the number of cells owned by each player are shown. These ownership tallies are utilised in the event of a tie-game, in order to ascertain a winner. This drastically reduced the number of tie-games and enabled a greater range of game strategies, furthering the utility of defensive cards and board domination.

If either player has any remaining unplaced cards at the end of the generation phase, the game reverts to the card placement phase. Once all cards, for both players, have been placed, the game then progresses to the end of round screen.

3.5.8.5 Game Loop: Round End

At the end of a round, a panel detailing the outcome of the round is shown over the game board. It is updated to reflect who won the round and how they won - either through traditional points scored or through a cell-ownership-count-based tie-break decision. A timer is used to allow players to absorb this information, and this timer value was increased, following initial informal user-feedback, to five seconds. A heart is detracted from the the losing player's lives, and the remaining number of rounds required for the winning player to win the entire game is shown on the round-end panel. To give the heart's disappearance an eye-catching visual element, the heart is at first increased in size and then decreases until it is no longer visible. These size transformations utilise time-based interpolation to ensure that the transition is as smooth as possible.

When the round end timer has depleted, two potential routes can then be taken. If a player has run out of hearts and has lost the game, then in this instance the game progresses to the game end screen. If both players have hearts remaining, the board is reset to an initial blank state, and all players' cards in their decks are reset to be available to be played again. Scores and living cell counts are reset to zero. The game then progresses to the Card Placement phase once again, and this cycle repeats until a player has won the game.

3.5.8.6 Game Loop: Game End or Game Quit

When a player has been declared the overall winner, having won three rounds, the play view is shifted to a menu with an updated text field declaring the winner. This screen is also the same screen utilised as an in-game menu, accessed by pressing the *M* key. Originally accessed using the more traditional *Escape* key, the WebGL build of Unity already uses this key for exiting full-screen mode. The options presented to the player are to start a *New Game* or to *Quit* to the main menu. Starting a new game will maintain the current game mode but reset the entire game and return the player to the *Card Selection* screen. In online multiplayer games, this requires a different approach and this is detailed in the online multiplayer game mode section. Quitting a game at any point in time will also reset the entire game back to its initial state.

3.5.9 Game Modes

There are three distinct game modes that players can choose from, listed in order of development: Local Multiplayer; Online Multiplayer; Single Player. All three game modes, for previously mentioned performance concerns, are executed within the same scene and it is the responsibility of the *Game Manager* to ensure that the correct order of execution occurs for each respective game mode. Whilst this approach has led to the code base requiring an increase in the amount of conditional branching, it offers significantly greater performance and has the additional benefit of being able to re-use code fragments as much as was possible.

3.5.9.1 Local Multiplayer

Local multiplayer games are used when two players wish to play against each other on the same computer. This was the first game mode developed as it facilitated debugging and included the majority of the core functionality required for subsequent game modes.

In local multiplayer, card selection occurs sequentially, with player one choosing their cards first. When the game begins the general layout of the game view is the same as in other game modes, however when player one has completed their turn the game view automatically shifts to show player two's deck. Visual prompts at the top of the game board notify players of whose turn it currently is. In line with all other game modes, player one's cells are denoted with blue cells and player two's cells are denoted with red cells. An expanded image of the game view is shown in Fig. 3.9, although it is important to note that at any one time only one of the player's decks will be visible.

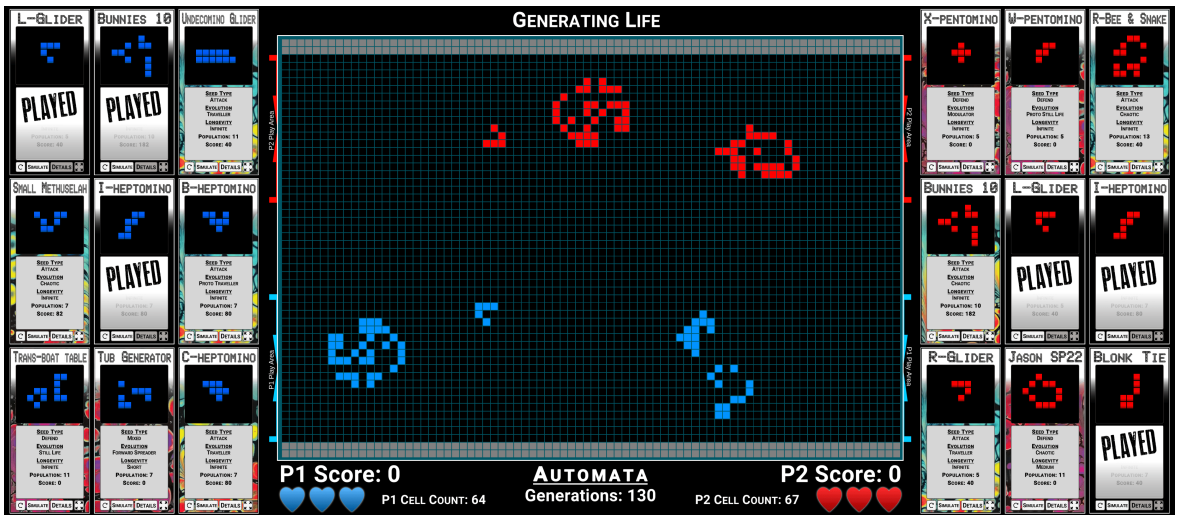

Figure 3.9: Full Game View - *Shows a board after the first round of card placements, with two gliders traversing the board towards the opponent's goal line.*

The key consideration when developing the local multiplayer game mode was the requirement for player two's seed placement preview to be rotated 180° in order for offensive cards to traverse in the correct direction (downwards). This was achieved by ensuring that the bit array produced by the base-64 string decoding was the correct length (i.e. the number of cells when multiplying the height of the seed by its width) and then reversing the order of the bits in the array. The result of this can be seen in Fig. 3.9 above with both player one and player two's gliders being rotations of each other. This functionality has wider usages across other game modes, and when manually analysing seeds in the event that they traversed in the wrong direction and required rotation before being designated as Attack cards.

When player one has completed placing cards on the board, the visual board is reverted to its initial state (the state prior to any placements by player one) to ensure that player two does not have an unfair advantage and to maintain the *Simultaneous Game* design of *Automata*.

3.5.9.2 Online Multiplayer

The online multiplayer game mode represented the greatest development challenge for the project, both in terms of technical solutions and synchronicity across game instances. The technical challenges were an inherent consequence of the decision to create a WebGL build of the Unity game. This decision had been taken to ensure ease of access for end-users, without any requirement on users to download and install the game in order to play it. The disadvantage that this created, however, was that Unity's in-built online multiplayer functionality requires at least one instance of the game to act as a host and to manage server-side operations. Due to security-related restrictions on web browsers, it is not possible to implement the same online-systems that are achievable with stand-alone Unity builds. As a result, WebSocket communications were the optimal solution to overcome these barriers and enable online multiplayer in web browsers. The *Native Web Socket* [40] package was used in combination with an AWS API that allowed users to connect to a DynamoDB endpoint where players would be connected to each other through storage of two universally unique identifiers (uuid), one for each player, on a single row of the DynamoDB table. When sending messages between clients, the table is queried for the sender's uuid and the message is then sent to the player whose uuid is stored in the same row. Each row, therefore, represents a game session that enables direct communication between players.

There are two forms of matchmaking that players can choose between for online multiplayer: open matchmaking, where players join the first available game session with at least one free space; and closed matchmaking, where players enter a session name which is used as the basis of a query of the DynamoDB table and, if a session containing that name is not found, then a new entry is created and subsequently another player may enter the game session by inputting a matching session name. In open matchmaking, the session name is set as "empty", and for this reason players are not allowed to use this keyword as a session name in closed matchmaking and a visual prompt will ask them to enter a suitable session name before continuing.

When a new online game is started, a WebSocket Service object is instantiated. Communications within the game loop make use of the *onMessage* API pathway which is connected to a Lambda function on AWS. The function takes a JSON object as input, containing an opcode and optional message body. The opcode is used for a conditional switch statement in the Lamda function and unique opcodes exist for: communicating player names; card selections; end of turn seed placements; the cards that a player has used; and starting a new game. Matchmaking is also triggered by sending either an open or closed matchmaking opcode, the latter additionally sends through the session name in the message body. When arrays need to be sent, such as in the case of the indices of the cards that a player has played, the array is first converted into an under-score delimited string and then this is split and parsed back to integers by the receiving game instance. Opcodes and message bodies are sent to the opponent through the API, triggering the appropriate in-game methods through the use of a switch statement.

If a player leaves, through closing the browser tab or by quitting from the in game menu, they will automatically be disconnected from the DynamoDB table. Each entry in the table has a *Game Status* attribute, and the Lambda function responsible for handling disconnects will query the table for the player's uuid and set the *Game Status* to *Closed*. The next time the remaining player sends a message to the API, they will be notified that their opponent has disconnected and, after five seconds, will be returned to the main menu. Game messages are sent through an invoked repeating method and as a result, this happens almost instantaneously.

3.5.9.3 Single Player

Priority had been given to the development of the local and online multiplayer game modes, as it was deemed that two human players would provide a more engaging and competitive user experience and would therefore generate more valuable feedback when it came to players describing how they felt that they could be strategic and adapt to their opponent's choices. The inclusion of a single player mode was to ensure that all users could have a game mode to engage with, regardless of whether they had an in-person or online partner to play against. Of the three game modes, it is the area that has the potential for the greatest level of future development and this will be discussed in more detail in sub-section 5.1.4.1.

The computer opponent (AI) uses randomisation to structure their choices during the game loop. Their initial card selections are entirely randomised, with the game already having the functionality in place to ensure that no player has duplicates of a card in their hand. On the first turn, the AI will place anywhere between one and three cards. On subsequent turns, they will place between zero and the minimum of: the number of available cards left in their hand; and three (the maximum a player can place on a given turn). This ensured that any while-based loop statements did not get stuck attempting to find an available card in the player's hand that did not exist. In order to reduce the likelihood of the AI repeatedly choosing to play zero cards, the lower limit was increased to one every one-in-three occurrences. This randomisation is shown in the Listing 3.9 below, and it is important to note that Unity's Random method uses an exclusive upper-limit when taking integers as its arguments. Card placement by the AI was also randomised, with a row selected from the available placement area (20 rows) and a column taken from those available (80). Card placement takes place within a *for loop*, incremented up to the number of cards that were randomly selected to be played.

```
1 private int PickNumberOfCards(int cardsLeft) {
2     if (isFirstTurn || Random.Range(0, 3) != 0) {
3         return Random.Range(1, Math.Min(cardsLeft + 1, 4));
4     }
5     return Random.Range(0, Math.Min(cardsLeft + 1, 4));
6 }
```

Listing 3.9: Randomly Selecting Number of Cards to Play

3.5.10 Tutorial: How to Play

The last choice for players to choose on the main menu is the game's *How to Play* section. A trade-off was required to be made between using a video tutorial, which would have significantly increased the build size of the game, and instructional text fields used to explain visual elements from the main game loop. Due to previously encountered loading issues for older devices, the instructional screens were chosen as the best starting point for the in-game tutorial. They have been expanded to include more information based on informal user-feedback, however consideration for how this area of the game could be improved upon is given in the Further Work chapter of this report. Images of each of the tutorial screens can be found in the Appendix.

The first screen introduces the aim of the game in *Automata*, detailing how players are seeking to score points against their opponent in a game that is best described as *Pong meets Tetris, with playing cards*. A general view of the game loop and the victory conditions are given. Font choices are in line with all other areas of the game to ensure continuity and small capitals have been used to ensure clarity and readability.

The second screen gives a breakdown of the playing cards, explaining what each section of the card represents and gives a short summary of the information provided on the card's details panel. The three main attributes that a card has are its type (Attack, Mixed, or Defend), its type of evolution (Traveller, Still Life or Chaotic), and its longevity (i.e. how long it will live without all cells dying, if it is not interacted with).

The final screen gives a more thorough explanation of the game loop. It seeks to equip players with all the information required to get started with the game. The formal user-evaluation will be used as the basis for refining and iterating on the tutorial to ensure that it allows players to feel familiar with the game's format and core loop before starting to play. Keyboard shortcuts are also given on the final page, and there are navigation buttons for users to move between the tutorial pages.

3.6 Web Application Development

3.6.1 App Requirements

There were three distinct content requirements for the web app: an interactive learning section, introducing CA and allowing users to engage with example *Life* seed configurations; hosting of the Unity WebGL build; in-site surveys on the learning section and the Unity game. The decision to have the entire project self-contained within the app was borne out of a desire to make user-evaluation as straight-forward as possible for end-users, whilst also enabling the project to have the potential for long-term development beyond the scope of the current project. Due to the aforementioned security restrictions associated with online multiplayer games and Unity WebGL builds, it also provided the opportunity to utilise AWS's AppSync features which would ensure that any API keys could be securely stored to reduce the potential for malicious activity. The site can be accessed here: *www.automata-game.co.uk*.

3.6.2 UI Design Choices and Layout

Consideration was given to the impact that different website themes have on the user, in particular the colour palette. Whilst dark mode themes have not been proven to improve readability or have significant impacts on eye-tracking [41], their usage and popularity has significantly increased over the last five years [42]. Given this trend, and the desire to make the general layout and style of the application to be as simple and easy to navigate as possible, a monochromatic gray-scale colour theme was chosen. The app's layout uses a top-of-page navigation panel for accessing each of the website sections, and a more detailed sidebar with each of the individual learning pages and their titles presented as a bullet point list.

The application uses CSS to ensure that it renders well on both mobile, tablet and PC. In mobile portrait mode, the side bar is hidden and the navigation bar is the sole means of moving to different sections of the site. This decision was taken to ensure ease of use, at the expense of having the same range of navigation options as are available on other platforms. For any images and *Life* examples, grid layouts are used and when viewing the site on mobile, the grid templates are updated to use multiple rows in place of multiple columns to reduce the requirement on users to have to side-scroll on individual elements of the main page. *Life* examples, such as on the Home page and the *Life Playground* page use a calculation based on the available screen width and height to determine how many cells should be in the grids that represent the *Life* lattice (List. 3.10). These will only update on page loads. This was achieved by having a reusable function that generated the *Life* grid and which takes row and column variables as part of their props input. Additional props included having interactive cells, having buttons to initiate or reset the simulation, having a label for the example and to determine whether or not the grid has an initial seed configuration.

```
let size = Math.round(
    42 * (window.innerWidth / 3000) + 1.3 * (window.innerHeight / 150)
);
let sRows = window.innerWidth > window.innerHeight ? size : size * 2;
let sCols = window.innerWidth > window.innerHeight ? size * 2 : size;
```

Listing 3.10: Determining the number of rows and columns for *Life* example grids

The survey pages were created using SurveyJS [43], an open-source JavaScript survey library. Survey question formats were, whenever possible, chosen to be Likert scale five-point rating questions. When more detailed response items were required, matrix multiple choice formats were used. For qualitative input on more general topics such as areas to improve, free text questions are included. A CSS style-sheet was used to ensure that the colour palette of the surveys was in line with the rest of the application. When navigating to the individual surveys, users are required to confirm that they have completed the consent form before continuing with the learning or game-based surveys. The consent form includes a downloadable PDF version of the information sheet, including full details of the study and contact information.

3.6.3 Learning Section Content

There are five core pages that comprise the learning section of the website. Each page includes increasingly complex information, building on the knowledge provided in previous pages. The content is aimed to be of a level suitable for sixth-form and first year undergraduate Computer Science students. Examples are given throughout to illustrate the information presented, including *Life* simulators, coded in JavaScript, YouTube videos showing bacterial colony growth in a Petri dish, links to art projects photographing the different forms that Conus seashells can take (mirroring the 1D Rule 30 CA pattern) and any wider suggested reading for individuals who wish to learn more on a specific topic.

The *Life* examples work up from simple one-dimensional examples, based on the *Hat Rule* example given in the *Stanford Encyclopedia of Philosophy* [44]. This particular example will produce new rows in a table for each new generation, up to a maximum of 17 rows. When a new row is created, the main page's scroll bar is set to the bottom-limit to ensure that the grid remains visible on screen. Once 17 rows have been created, the top row is popped from the table's array data structure and a new row is pushed to the end. This maintains a maximum size for the grid, whilst also enabling the simulation to continue generating new rows.

The next page introduces *Conway's Game of Life* and its fundamental rule set, the *Moore Neighbourhood* count with a labelled *Life* grid for illustrative purposes, and the notion of grids wrapping from left-to-right as this is used in the *Automata game*. Following this, the three main categories of seed configurations (Oscillators, Spaceships and Still Lifes) are introduced with example life grids. The JSON document that is used in *Automata* has been re-used in the website and the dataset is queried based on the names of the seeds, providing the *Life* instances with all of the data required to create the example simulations. The same decoding of the Base-64 string is utilised to create the bit patterns that populate the grids with the seed configurations. The still life examples are interactive and users are prompted to add or remove cells to explore the delicate balance that enables still lifes to maintain their form.

The final learning page gives details from Andrew Ilachinksi's book [45] on the four areas that make CA useful in academic study:

- As powerful computation systems.
- As simulations of dynamic systems that have a finite size.
- As models of pattern formation and complexity.
- As original models of fundamental physics.

Whilst this page is solely informational, its purpose is to create a sense of the real-world applicability of CA and how they are used in research for fields ranging from physics to philosophy. This page is followed by a full-screen interactive *Life* playground for users to create their own configurations and see how they evolve, decay, or remain in stasis over time.

Chapter 4

Project Evaluation

User evaluation was carried out through means of two surveys completed within a single session, one relating to the learning section of the web application and one relating to the *Automata* game. Participants were informed of what the study involved and the reason behind the data collection process prior to giving consent through an online form that digitised the blanket ethics form provided by the University. A downloadable PDF version of the information sheet was available at the top of the Consent form. Users took part in the evaluation through the online web application, and they included those who are currently studying Computer Science and those who have never studied it. This ensured that users possessed a wide range of domain-specific knowledge to better support the evaluation of the project as a learning tool. Where possible, users were invited to play the game with their partners and, due to the online features of the game, this was possible for partners who currently live in different countries. This suggestion to users was made informally, however it was given in the hope that this would drive a sense of competitiveness between the individuals playing the game. The following sections give full details of the questions included in the user-evaluation, alongside an overview of the design and results of the surveys as well as a critical evaluation of the project and its findings.

4.1 Survey Design

Users were invited to evaluate the project with an introductory statement giving details of the process and the constituent parts of the website:

> *I'd like to invite you to take part in the evaluation of my MSc project. The project is comprised of a website with a section on learning about cellular automata and a game, made in Unity, based on Conway's Game of Life. If you've never encountered either Cellular Automata or Game of Life before, don't worry - everything is introduced on the site. If you are happy to take part, I would really appreciate it if you started by completing a consent form, which can be found under the survey tab of the site, and then there are two parts to complete: the learning tab to read the information given and play with the interactive examples, followed by the learning section survey; and the game, found under the game tab, followed by the game survey. The game can be played on laptops and desktops, and has both single and multiplayer game modes. You can play for as little or as long as you would like! If you have any queries about the site, the game, or how the survey data will be used, please don't hesitate to get in touch with me.*

Consent was given using the blanket ethics form provided by the University of Bristol, adapted to provide the full information and set of questions in an online form (as detailed in Section 3.6.2). Users could not progress to either the Learning survey or the Game survey unless they had confirmed that they had completed the Consent form. All form data is stored securely in an AWS DynamoDB table, accessed using an API key with only the *Write* permission set to ensure that users could not read or update any table entries that have already been submitted. SurveyJS enables the output of the survey to be in JSON format and a GraphQL schema was created for each of the surveys so that the survey outputs could be successfully uploaded to the tables. Both the Learning and Game surveys are multi-page surveys and, based on informal user-feedback, the user's current page is tracked at the top of the form to allow them to have a sense of the remaining time requirements of the survey.

CSS stylesheets were used to ensure that survey questions could be easily read and that the colour scheme utilised was in keeping with the rest of the web application. The surveys could be accessed through the top panel of the site and therefore were well sign-posted for users.

4.1.1 Learning Survey

There were three key parts to this survey: the overall design and usability of the website; an assessment of the content of the learning pages; a free text question for reporting of any general issues or bugs that users experienced. A total of 15 users completed this survey.

4.1.1.1 Website Design

Users were asked to report the type of device they used to access the website, before being asked the extent to which they agreed with statements on the general design of the site, the colour scheme used, the ease of navigation offered by the site's layout, the size of interactive elements and how well it rendered on their browser. A final 5-point rating was then collected on the overall usability of the website.

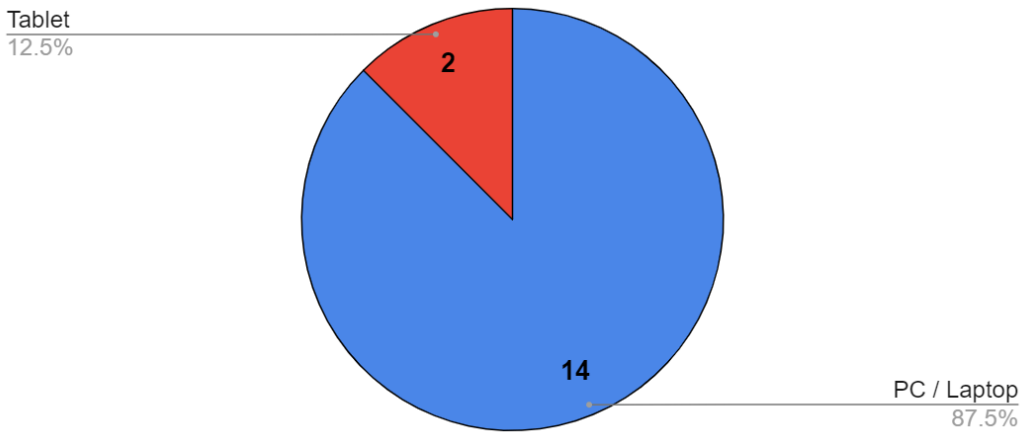

Figure 4.1: Website Device Usage Results

The majority of users accessed the website on a PC or laptop. This is most likely due to the fact that the Unity WebGL build of the game is not supported on mobile, and this message was shared with users as part of the user-evaluation invitation. It was further reinforced when accessing the site, as the page of the application for the game will conditionally load, based on a media query of the device being used to access the page, either the Unity game or a prompt to access the site on a laptop or PC. The number of users who accessed the site on a tablet was not sufficient for any statistical comparisons to be reliably drawn.

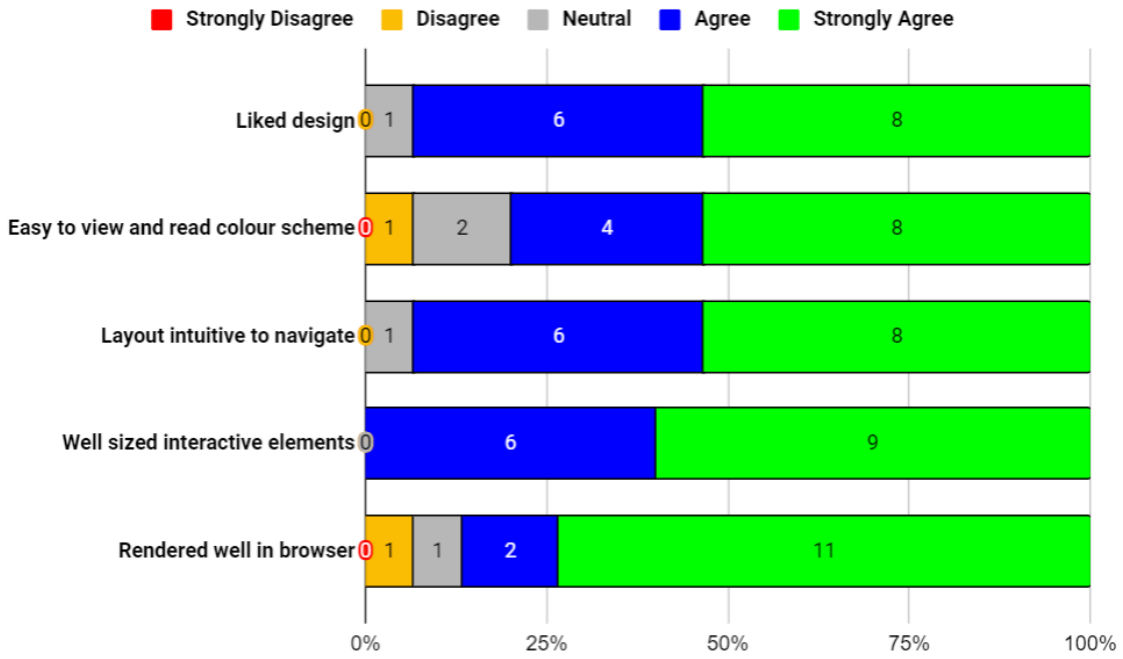

Figure 4.2: Website Design Survey Results - *Green always denotes the most positive response*

The vast majority (14 out of 15) users liked the design of the website, and found it intuitive to navigate and rated its usability highly. All users felt that the interactive elements were well sized. One user disagreed on both the colour scheme ease of reading question and the rendering question. To further illustrate, feedback given in the free text question in the subsequent page noted that "*some of the next / previous buttons were not styled, and the black text on grey background isn't great*". This was further supported by another user, who felt that "*the stark black and white on some of the pages was quite tiring to read*". It could, therefore, be of value to include light and dark modes for users to choose between based on personal preference to create a more individually tailored user experience.

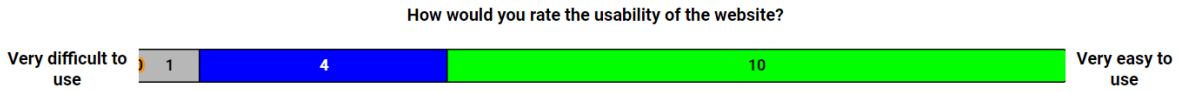

Figure 4.3: Website Usability Survey Results

4.1.1.2 Learning Content

This section of the survey related specifically to the learning pages, rather than the wider website's design and layout. Users were asked if they had any prior knowledge or experience of CA or *Life*, whether they found the content easy to understand and interesting, whether it was presented clearly and concisely, whether the *Life* examples and interactive playground were useful to support understanding and whether there were sufficient examples of the use of CA in academic research. Two distinct questions explored the extent to which users' understanding of how CA work improved, and if their understanding of why it is used improved. Free text input fields were then given for users to detail anything they enjoyed or would like to see improved on the learning pages. Seven users had heard of *Life*, two had heard of CA, and eight had not heard of either *Life* or CA before taking part in the study.

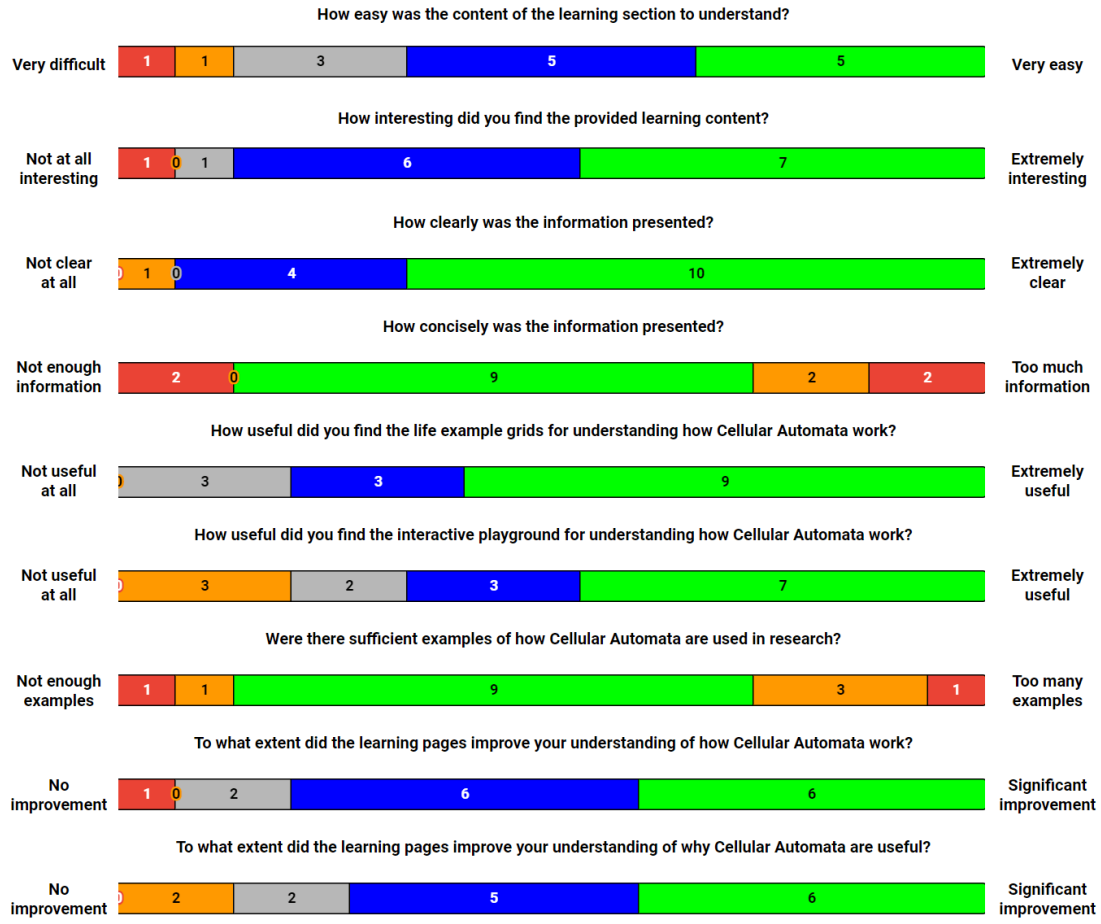

Figure 4.4: Learning Content Survey Results

The findings were on the whole positive, with the majority of users finding the information to be clearly presented, interesting and, to a lesser extent, easy to understand. The playground was deemed to be useful by two-thirds of users, and this is therefore a notable area for development.

One user suggested the inclusion of an option to have a two-player version of the playground, akin to the game board found in *Automata*, to enable users to have the ability to create in-game scenarios and see how they turn out: "*I think many users would like to know how cells work when there are 2 types of cells around it (Your cells and your opponent's)*". This was supported by another user: "*Animated demonstrations / walk through of game play.*".

Users were either happy with the quantity of content provided in the learning section and the number of examples given of the use of CA in academic research, or were polarized between wanting more or less information. This was noted to be of particular importance in the academic research section with one user stating that "*The motivation section had a little too much text to read, which can be a tad hard to focus on without any images or examples to help*". Another user noted that more information on three-dimensional CA with real world examples would be a welcome addition to this section.

The following quotes are each from individual users when asked what they had liked about the learning section and its content:

- *"They were concise and the visual representations / animations helped me understand the concept quickly."*
- *"The interactive still life examples were particularly good at it was really easy to engage with by giving that start point - it was actually a pleasant surprise that they were interactive!"*
- *"Very interesting factual material."*
- *"I found the shell particularly fascinating. Excellent range of examples that increased my understanding."*
- *"Explained in an accessible format."*
- *"Inclusion of examples found in the natural world."*
- *"Really well presented, clear, just the right amount of images."*
- *"The range of examples quoted and the explanation of how a few simple rules can lead to complex patterns/constructions."*
- *"The interactive examples."*
- *"The interactive examples and playground is really good for player to understand the game."*

Users on the whole did not report any significant bugs or issues with the site, however one user did request that the game section was made more obvious as they had thought that the interactive playground was the main *Automata* game. One bug was identified during informal user-evaluations, relating to the playground's reset button and this was immediately fixed prior to any further user-interaction.

4.1.2 Game Survey

This survey covered the *Automata* game and was comprised of four main sections: general functionality, utility of tutorial, and chosen game mode; playing card design; gameplay, including understanding of rules, application of strategy, enjoyment, likelihood to replay; general bugs and issues. Question formats included four-point descriptive multiple choice, chosen over the less descriptive 5-point rating format to remove the neutral aspect for understanding-related questions, free-text responses, and, where applicable, 5-point Likert-scale ratings. A total of 16 users completed this survey.

4.1.2.1 Introductory Questions

Fourteen respondents experienced the Unity WebGL game loading and playing well in their browser, whilst one reported that it loaded and was acceptable and another user noted that it loaded but performed slowly on their computer. Users choice of game mode was evaluated using a multiple-input checkbox question, and the majority of users played in the Single Player mode (15) whilst one played in local multiplayer and three played online multiplayer games.

Most users (13) found the *How to Play* section useful for introducing them to the game's mechanics and game loop, however three users gave this a rating of 2 out of 5 for usefulness. Free text feedback on the tutorial detailed that there were "*clear instructions, easy to follow*" (2 similar comments) and that it was "*clearly laid out*". Suggestions for improvements included having "*some tips of tactics*", "*an animated, staged work through of gameplay*" and "*an example simulated game*". Based on this user-feedback, the tutorial is, as detailed in Chapter 5, a key area for further development to ensure players have the information needed to play effectively.

4.1.2.2 Playing Cards

Likert-scale ratings were used to assess users' views of the playing cards, including the general design, ability to simulate the *Life* preview, ease of reading the card and ease of viewing the card's colour scheme. Findings are given below in Figure 4.5.

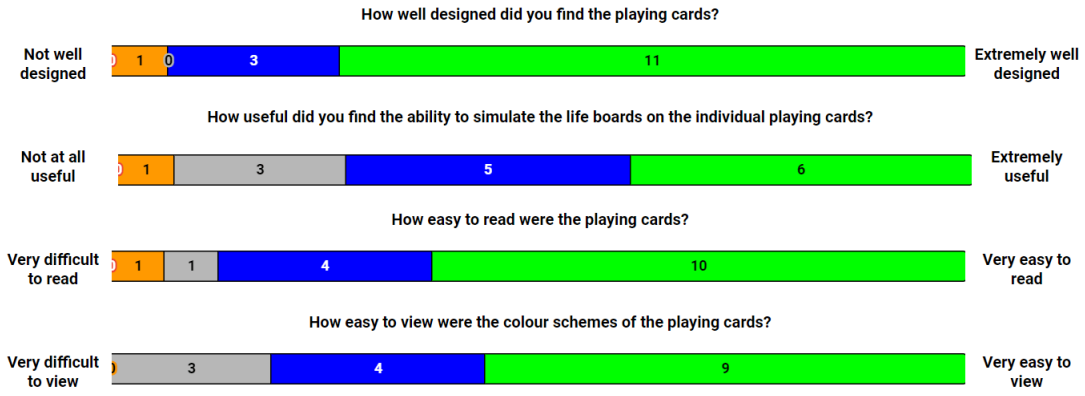

Figure 4.5: Playing Card Survey Results

The majority of users found the cards to be well designed, however one user felt that they could be improved upon. There were no illustrative text-based comments given in subsequent input-fields to give greater detail to this, however based on other responses by the user this was likely linked to text being difficult to read, particularly on smaller laptop screens as was noted in direct informal feedback. Whilst some users (6) found the *Life* preview simulation useful, more could be done to improve upon this. As noted in the Further Work chapter of this report, being able to magnify a card would give the preview significantly more screen space to be able to give a larger and more useful simulation.

4.1.2.3 Understanding Automata's Rules

Labelled multiple choice questions were used to assess rule understanding, with labels ranging from users having *no understanding* to having a *strong understanding* of the stated rule. Areas where the game performed strongly related to the game-loop specific rules, such as the number of cards that could be played and when a round or game ends. Improvements could be made to players' understanding of the aim of the game, how to play and where the player is able to place their cards. Whilst a custom cursor replaces the standard cursor, showing a large red no entry sign, further user-testing could be carried out to ensure that this works as intended on a range of different devices. Other improvements could be achieved through an improved tutorial.

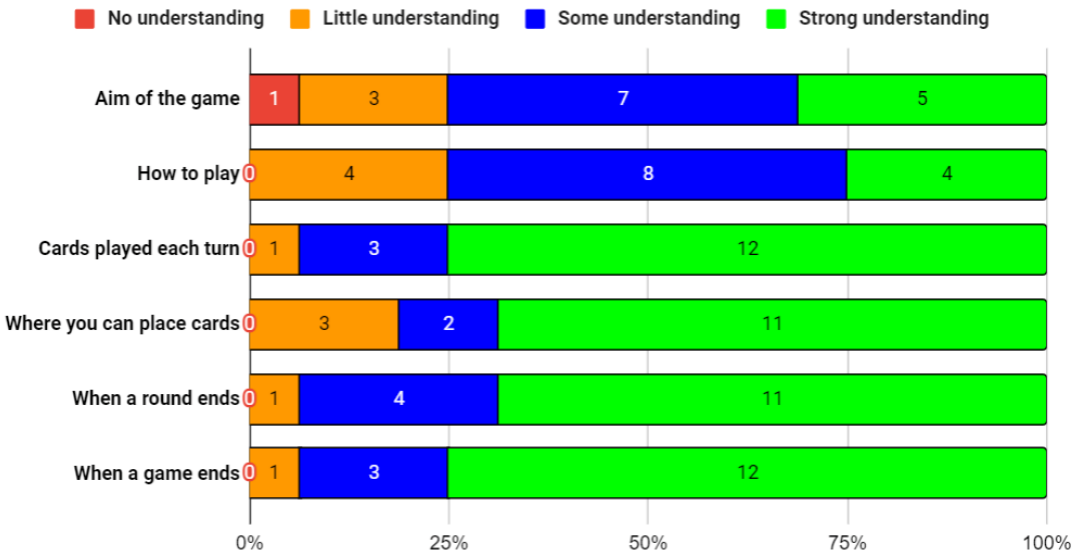

Figure 4.6: Game Rules Survey Results

The next set of gameplay specific questions, using a 5-point Likert scale, related to: players' ability to act strategically; whether continued playing and experience would make them a better player; how predictable the game was as it progressed; overall enjoyment; likelihood to replay; and the supporting of understanding of CA developed through playing *Automata*. The findings for this section of the survey are given in Figure 4.7.

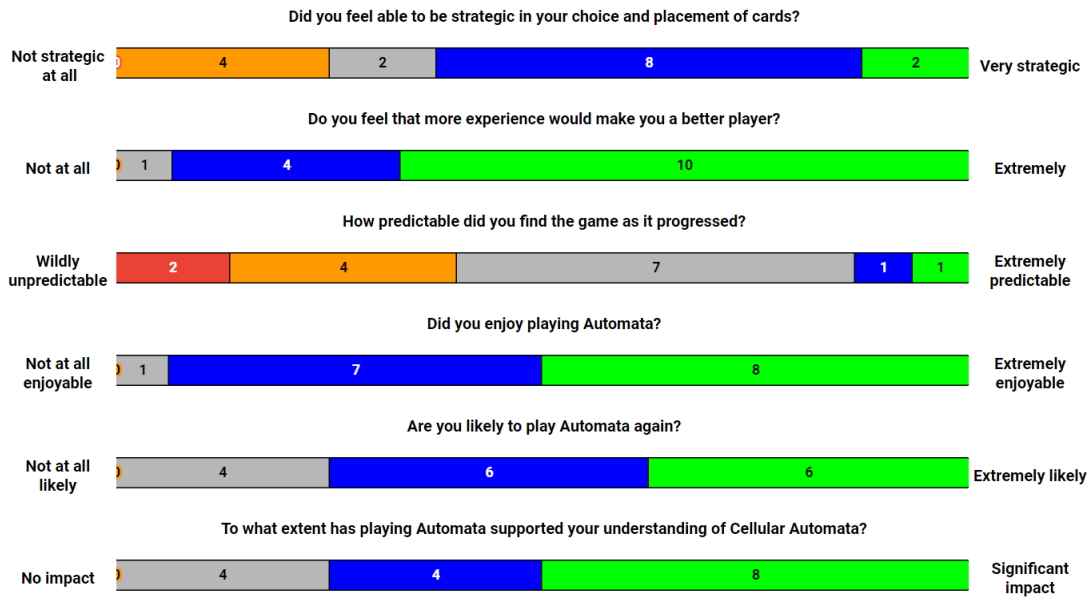

Figure 4.7: Gameplay Survey Results

Whilst some players felt that they were not able to be strategic in their choices, the strong weighting towards users feeling that continued playing of *Automata* would lead to being a better player is a positive reflection on the intention for the game to be easy to understand but challenging to master. Players did report a significant degree of unpredictability in the game, and refinements to the learning section's playground to introduce a two-player setup would support players in this respect. Users overall enjoyed playing *Automata*, were likely to play the game again and felt that it supported their understanding of how CA work.

In free text fields, players found that chaotic cards and travellers were the most useful cards. Gliders were reported to be the most predictable and easy to grasp Attack cards. Still Life cards were useful for defending the most likely line of attack. Users found that the least valuable cards tended to be Defend cards, and that some chaotic cards had a tendency to die out quickly. One card in particular, the C-Heptomino card, appeared to travel in the wrong direction and therefore could be well suited to be rotated to ensure it works well as an Attack card.

Initial user-feedback identified one bug, where a UI element was obscuring a button and this was fixed and the online version of the game updated to reflect this for other users. Some users experienced a wide variation in the computer opponent's difficulty, with it scoring well on some rounds but not others. One user felt that having the option to *undo* a card placement would be a valuable addition and also having the ability to choose how many rounds to play to enable users to play shorter games. Positive user feedback included:

"*Enjoyable, informative game that enhanced my understanding.*"
"*Very interesting concept, demonstrated by an engaging game.*"

4.2 Critical Evaluation

The overall reception of the project, based on the user-feedback received, has been generally positive. The design of the web application in respect of its usability, ease of navigation and user-interactions all scored highly in survey responses. The learning content of the site could have been further expanded, with more introductory pages to ensure that users had a strong baseline of information on which to build from. Examples of how CA are used in academic research were, on reflection, offered in an overly information-focused manner and greater utilisation of embedded videos and images could have been valuable tools to bring these research areas to life. The omission of three-dimensional CA, as noted by one user, was an oversight that could have driven greater levels of interest and also been linked to wider usages of CA in other games, such as in Minecraft where players have created 3D CA simulators within the engine.

User feedback included a satisfactory number of participants, representing a good degree of engagement with the project, however expert user feedback would have been a welcome addition. Two experts, both educators, were approached to take part in interviews on how the products might be used in educational settings, however, due to school summer holidays, their involvement could not be secured. Furthermore, interaction between the surveys and the game instance users had played could have introduced more responsive feedback elements, whereby players' decision-making at specific stages in a game could have been reflected upon.

The development of the game, and in particular the process of sequencing development to ensure that the functionality required for the next stages of implementation were in place, was successful. Throughout the cycle, time allocated to tasks was generally adhered to and the overarching vision for the game and the website were achieved. The online game mode was a highly challenging aspect of development and its prioritisation, in hindsight and on reflection of the distribution of users' choice of game modes to play, has not returned the same value for users who took part in the evaluation process. It is, however, a valuable part of the game for any future work that is carried out and has created a platform for which the game can be easily utilised in both classroom settings and spread more widely with online communities that have an active interest in CA.

The initial aims of creating a robust and bug-free user experience have been validated through user-feedback, and this stands true for both the web application and the game. Issues that were identified during user evaluation were promptly corrected and updated versions of the site and game were quickly deployed. The more specific intent to create a strategic game based on the rules of CA was more mixed, and whilst players did feel that their competence and ability to play strategically would develop with increased time spent playing, this aspect of the project would have benefited from longitudinal user-evaluation. The combination of the web application, as an introduction to CA, linked with the game has been successful in being seen as an educational platform that is enjoyable to play and improves users' understanding of CA and how they work. In this respect, the educational aims of the project were successful.

Chapter 5

Further Work

This chapter explores the potential for new avenues of development for the *Automata* game, relating to the user-interface, the in-game tutorial, online interactions and the game's mechanics, as well as the web application and its potential wider utility. Ideas for further projects that could continue the work carried out in this project, or that could explore new ways of implementing similar systems in novel ways are also discussed.

5.1 Expanding Automata

5.1.1 User Interface

The element of the game's UI that would benefit most from further development, to improve readability and utility, is the playing card's design and details panel. Some users reported that on certain device screens the text was difficult to read and that, whilst this was alleviated when using full-screen mode, this was an area that could benefit from additional consideration. One potential avenue for improving this experience for players, whilst still maintaining the same level of detail in the active game view, would be to have a magnification feature for the playing cards. This could be implemented as either a hover-over function, utilising the fact that the entire card panel has a *button* component with in-built hover-related functionality, or through a clickable button to magnify the selected card. In both cases, the magnified card would be best placed over the game board so that alternative cards could be selected for magnification without obscuring any of the deck from the player's view. This could be executed by having an additional, larger playing card *prefab* that is always present in the scene but is deactivated by default. When the player clicks to magnify a card, the selected card's *Card Manager* would be able to send the index of the seed to the magnified card's *Card Manager*, ensuring that the correct details and *Life* preview are showing before then activating the *prefab*.

The contents of the details panel could also be improved so that in the event of a player expanding the details panel, additional information was shown relating to the card. This could include a developer-written description of the card and its utility in the game. It could also identify the individual who discovered the seed configuration or include other information that could be collated from the *Life Wiki* website, such as the commonality of the configuration or association to other seeds.

5.1.2 Tutorial

The game's tutorial is an area that could be improved upon to support players to have a greater understanding of the game's mechanics, game loop and potential gameplay strategies. Some users suggested that this could be achieved through a video tutorial, and whilst this might cause complications on build size, as previously noted, there is the potential to animate the UI to simulate a tutorial video using only the elements already used in the game's scene. This would require no additional memory, but would require careful planning to ensure that the simulated gameplay is valuable and informative to view. One additional way to ensure that users interact with the in-game tutorials would be to ask players, when first selecting a game mode, if they have ever played *Automata* before and if not then to take them to the tutorial section of the game.

5.1.3 Online Interaction

There are several aspects of the game's online interaction which have significant potential for further development. One of the most simple aspects would be the inclusion of an in-game chat function. This would be able to utilise the *Native Web Socket* messaging system, and would require the creation of a new opcode that could be uniquely processed in the AWS Lambda function and receiving game-instance. It could be activated and deactivated through the use of the space-bar, with a chat box expanding from the bottom edge of the screen upwards using interpolation to ensure a smooth transition. In order to ensure that this key-binding did not interfere with users when they are typing a message, a Boolean value would need to be set in the script that handles the show / hide functionality of the bar and a public setter method would need to be called when the user's focus is on the input field.

Additional online functionality could be focused more on the game's community through the use of a high-score table, storing the name and score of the individuals who have secured the highest round-end point tally. This could be achieved by sending a new message to the *onMessage* Lambda function on AWS with an opcode denoting the message as a *high-score* query. The functionality for querying the DynamoDB table, removing elements that have been superseded and writing the new elements that have entered the current high-scores would ensure that write permissions are not extended to the Unity game.

A final consideration for online interactivity would be the inclusion of a seed-creator on the web application which could be used to update a DynamoDB table that holds the existing seed list, written from the current JSON file, with new entries. On loading the Unity game, through the use of the *react-unity-webgl* interface the community version of the seed list could be requested and used in lieu of the current set of seed cards that the game uses. Whilst there would need to be a degree of restrictions on cards, particularly relating to width and height, this would improve the levels of creativity and exploration that the game and application together would offer to users.

5.1.4 Game Mechanics

5.1.4.1 Single Player Opponent

The computer opponent that currently features in *Automata* is entirely based on randomisation. There are several areas where this could be improved upon to create a more challenging single player experience. The way in which the computer selects cards would benefit from having a requirement for the computer to select at least four Attack Cards. This is especially significant due to the fact that, across the entire deck of seed cards, there is a weighting towards Defend cards and therefore this makes the computer opponent more likely to be consequently playing a more defensive game. Another relatively simple addition to the opponent's strategy would be to make the computer more responsive to the choices that the human player has taken. For example, if on the previous turn the human player has used two Attack cards, the computer opponent could choose to play more defensively on the subsequent turn. Alternatively, if the human player has not yet played any defensive cards, the computer opponent could be made to be more likely to choose several Attack cards as a means of launching an offensive.

Card placement is also an area that could benefit from analysis of the game board prior to selection. Whilst remaining randomised, limitations on the bounds of this randomisation could be created based on the numbers of living cells for a given sub-section of the game board. To ensure speed of execution, this could be initiated by splitting the placement area in half and determining which section has the lowest living cell count. By using a threshold ratio between the number of cells in each of the two sections, the computer would either place a card within the least populated section if the threshold had been reached or would further subdivide the least populated section and continue to loop through this process until either the threshold is reached or a pre-determined section size lower-limit had been reached. This would ensure that, in the event that both sides of the board are reasonably evenly populated with only a small variation in cell count, that placement can be more specifically targeted whilst also ensuring that no unnecessary iterations occur if a viable section has already been identified.

A new approach to the computer opponent would be to implement the *Monte Carlo Tree Search Algorithm* to replace the current randomisation as has been deployed for other board games, such as *Chess* or *Go*, to create intuitive and challenging opponents. It has also been used for grand strategy games such as *Total War: Rome II* [46]. Implementation would require a significant amount of development work and would need to take performance into consideration, especially given the range of potential options open to a player when determining which card to place and where. The speed of this computation could be improved by using the aforementioned subdividing of the placement area to reduce the scope of the search. The key principles of the search algorithm are analysing and exploring the most advantageous moves and utilising random sampling to expand the search area. The benefit of face-up cards is that there is sufficient information available to the computer opponent to run simulations of how moves might play out over one or more rounds, whilst also accounting for the human player's potential choices.

5.1.4.2 Challenge Modes

The selection of game mode's could be expanded beyond the current two-player format and this could have significant potential for the educational aspects of the game. By introducing challenge modes where the player is attempting to use seed cards to destroy current configurations or to capture sections of the board, whether individual cells or larger areas, this could help to improve the level of familiarity that players have with the seed cards. Restrictions on which cards the player has access to to complete a challenge would facilitate this. Challenges would need to have clear and easily validated victory conditions in order for this to be a straightforward avenue for development. Challenges could also interact with previously mentioned ideas for further online interactions by storing high-scores of players who completed a challenge whilst using the least number of playing cards or with the lowest number of generations simulated.

5.1.4.3 Automata's Wider World

On a significantly wider scope, the *Automata* card game could exist as a core mini-game within a wider role-playing game context. A key example of this type of implementation would be the *Pokémon* Nintendo games. By equipping players with a limited starting deck, from which they are able to choose 9 cards, they would then engage in card battles with computer opponents and receive new cards as rewards for victory. By gradually introducing players to an increasing number of cards, it enables players to build familiarity with each card. Individual cards could also have an experience rank which could relate to being able to place multiple instances of a card on the board. Online matches could be a central feature, potentially expanding to tournaments. Development could also focus on building the game for a wider range of devices, given the restrictions of Unity WebGL for mobiles, meaning that players could access the game on smart phones through an app and on handheld gaming devices like the Nintendo Switch.

5.1.4.4 Gameplay Analysis

Given the game's pre-existing framework for connecting with DynamoDB tables, and given a sufficiently large sample size to draw meaningful data, data collection could take place for each game played in order to track: board states at the start of each generation round; cards played by players, with associated meta-data of whether they won the round and how many points they scored; cards that are regularly discarded during the card selection phase of the game; time spent choosing cards; time spent deciding moves; time spent awaiting matchmaking; numbers of cards placed on each round; number of games played within a single session.

Board states can be encoded once again as Base-64 strings and stored as an entity in a table. Only the state of the board at the start of each generation round is required to be known as that will determine the state of the board at the end of the generation round. Analysis could then be carried out to ascertain which cells, or subsections of the board, are most likely to be populated at the start of the generation round to determine the most common areas for players

to place their cards on the board. This data could be used then to make a more challenging computer opponent. Dither time could also be used to introduce timers to the game, set at a percentile that covers the majority of players whilst encouraging those who take longer to make their choices to speed up their decision-making to ensure a smoother online experience. Cards that are regularly discarded and replaced could be removed from the game and replaced with some of the alternative seed cards available, and this refinement process could be an active and ongoing endeavour.

5.2 Expanding the Web Application

Interaction between the online aspect of *Automata* and the web application could be improved, with users being able to view how active the online matchmaking lobby is and the times of day when it tends to be most active. Game data that is collected through in-game analysis and stored in DynamoDB tables could also be publicly presented, with the ability to watch replays of matches that have been played or to view the most popular (or least popular) playing cards. If the website included a seed-creator, then community creations could have a system whereby they could be shared and voted for.

The web application should not seek to be a competition to the pre-existing and extensive online resource for *Conway's Game of Life*. It could, however, be further expanded to include new learning content sections. These could go further into areas such as the philosophical implications and thought experiments associated with *Life*, with an additional workshop on the concept of *Emergence*. It should be noted that whilst there have been effective learning outcomes for workshops on this topic, the subject matter does require a level of understanding that may be less universally accessible [47].

5.3 Beyond Automata

This project has created a strong foundation for larger areas of work that go beyond the scope of iterative development. The entire game could be re-created in a variety of formats, whether this was an augmented reality mobile game that uses a tabletop as the game board whilst adhering to the same set of rules as laid out in *Automata*, or by creating a far more interactive, physical table top game that more closely echoes the vision set out in the aforementioned quotation from David Brin's novel *Glory Season*.

Greater exploration of the notions of intrinsic and procedural learning could also be carried out, through a project that created a range of different games that sought to assess the potential learning outcomes that could be achieved through embedding of academic goals in the games' mechanics. By evaluating the impact of different subject matter within the same game, and different games used for the same subject matter, comparisons could be drawn to evaluate the extent to which it is the medium or the message that has the most significant impact on learning in educational games and how these two attributes of a game interact.

Chapter 6

Conclusion

This project has resulted in the successful creation of a Unity WebGL game, featuring single, local and online multi-player modes, embedded within a React web application that also features an educational workshop on CA that is self-authored and tailored to the information requirements of the game. The products created were reported in user-evaluation to be robust and reliable to use. This chapter reflects on each of the respective aspects of the project with respect to the initial requirements and aims set out for them. It closes with a reflection on the project and the contributions that it has offered.

6.1 The Web Application

The primary aim of the website was to create a platform that offered an introduction to the key principles that underpin CA and to give users a solid foundation for engaging with the more advanced content available on the site and to have an understanding of the mechanics at work in the *Automata* game. It was necessary to introduce a certain degree of information to users in order for the procedural learning elements of the game to be effective, but aimed to not over-complicate the material by progressing towards the more complex aspects of CA. User evaluations provided evidence that, for a notable majority of users, the content provided was easy to understand and was interesting to read. Further development of the interactive playground page would provide additional preparation for users to engage with the game and have a better understanding of what to expect when playing.

The site's user-interface was almost universally well received by those who took part in the evaluation process, and informal developer testing through the use of a web browser's developer tools showed the site to be well presented on a wide range of screen formats. The adaptive nature of many of the components in the React application ensured that the site was rendered appropriately and *Life* grid examples were correctly proportioned to fit within users' devices' screens. This combined use of JavaScript and CSS was successful and has led to a viable product that has the potential to be expanded upon and further developed.

The use of AWS's Amplify to host the site enabled the utilisation of other AWS applications such as DynamoDB and Lambda functionality, which allowed a solution for *Automata's* online multiplayer mode to be found and executed successfully. It also ensured security with respect to users engaging with the online game mode as well as for collection of user evaluation data. Offering the entire project within a single domain space has facilitated users to interact with the project and have easy access to each of its constituent parts.

6.2 Automata - The Game

The development of the game and the resulting product can be seen as being a success. The initial vision for the game has been enacted and the user experience when playing is reported to be engaging, rewarding and competitive. The three different game modes have been provided in a robust and memory efficient format by deploying them within a single Unity game scene. The seed analysis functionality of the *Life* simulator enabled an automated refinement of the large bank of seeds available, reducing the total from the initial three-thousand two-hundred. The work that went in to ensuring that seeds and boards could be encoded and decoded from Base-64 strings reliably enabled the wider use of the seed bank in the website and its interactive *Life* examples. It offers the potential for up-scaling and community input in the future and ensures a common format across all aspects of the project.

The game plays fluidly when playing against the computer or when playing online against a human opponent. User evaluation has shown it to be fun and engaging and has cemented the learning that took place when reading the website. The online game mode made good use of DynamoDB to offer a cheap and effective solution to game session hosting and enabled both open and closed matchmaking so that users could play with random opponents or friends alike. The technical challenges associated with creating multiplayer games within a WebGL build were demonstrably met and overcome, with the resulting gameplay feeling both seamless and reliable.

6.3 Project Contributions

The key contributions offered by this project are insights into how the mechanics of a game can be utilised to support procedural learning by embedding learning outcomes in the actions that players carry out within a game. It shows that, when equipped with the foundational knowledge required to interact with and understand a system, there is the potential to create fascinating games that use these systems as their basis and where the decision-making that players undergo can be used to embed the knowledge that they have previously acquired. It offers a strong platform for the development of this concept to further take advantage of the potential of CA in educational settings, and gives credence to the idea of using this format to introduce these concepts to students who are studying Computer Science as a means of teaching them the underlying concepts in an exciting and engaging way. It supports the notion that, as is so often the case in programming and computing, equipping individuals with the ability to understand superficially simple systems enables them to be self-directed and intrinsically-motivated in their discovery and exploration of the complexity hidden within.

Appendix A

Appendix A

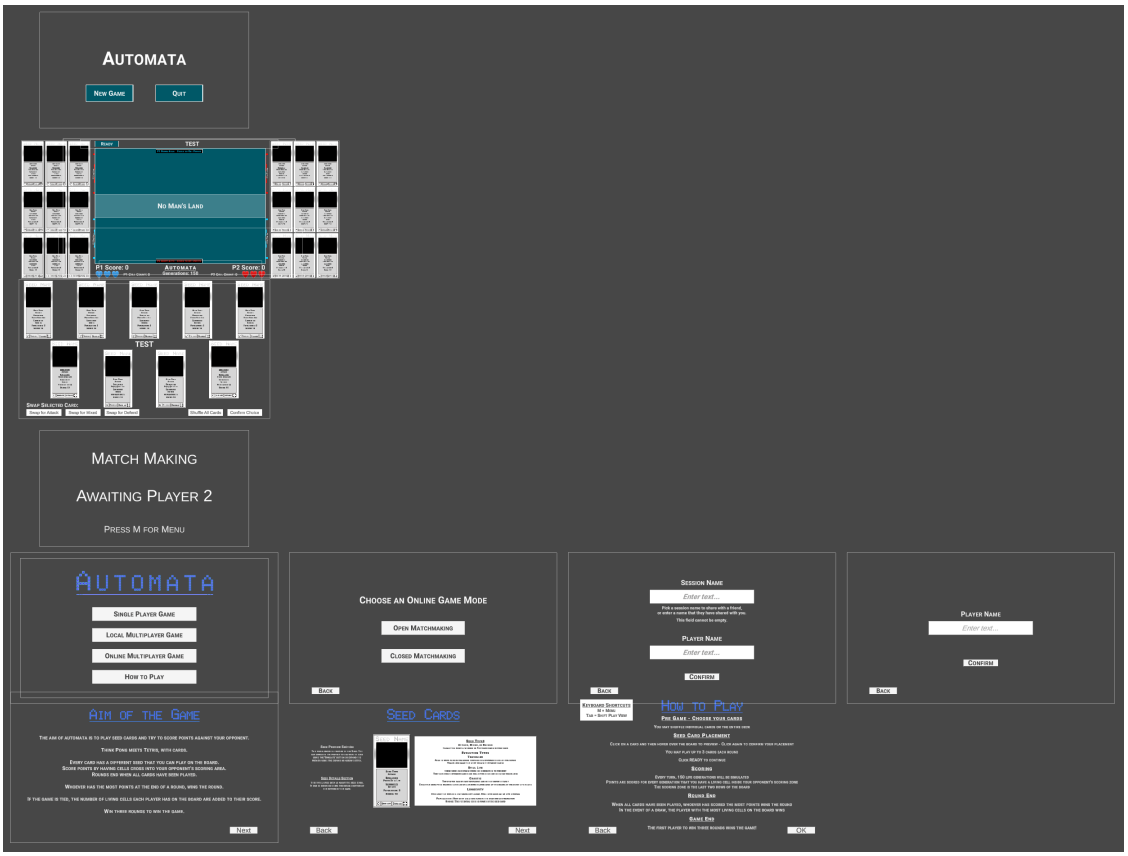

Figure A.1: Unity Scene Management

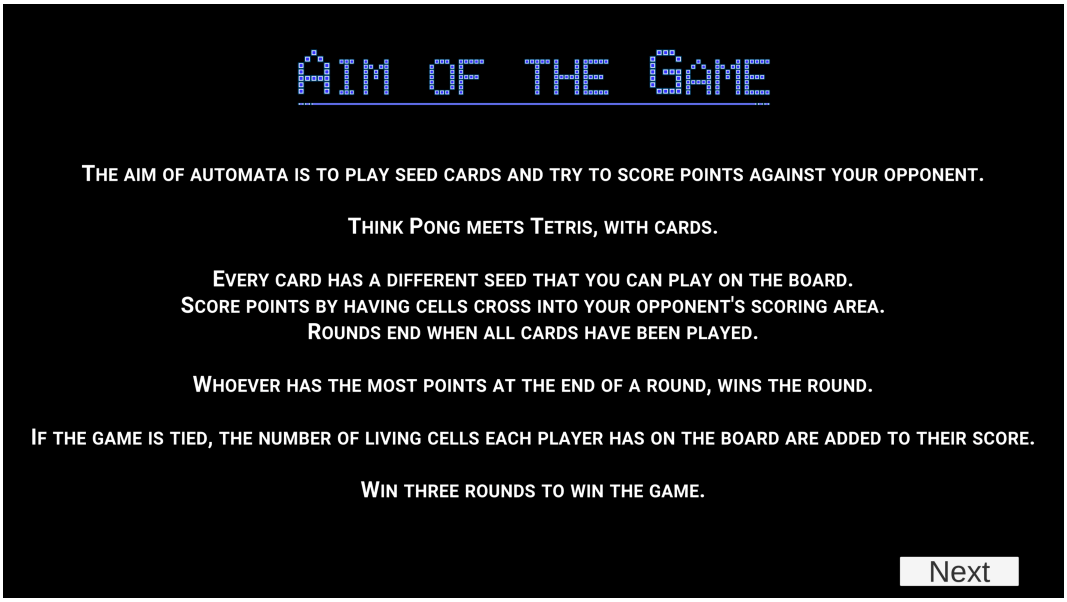

Figure A.2: In-game Tutorial Page One

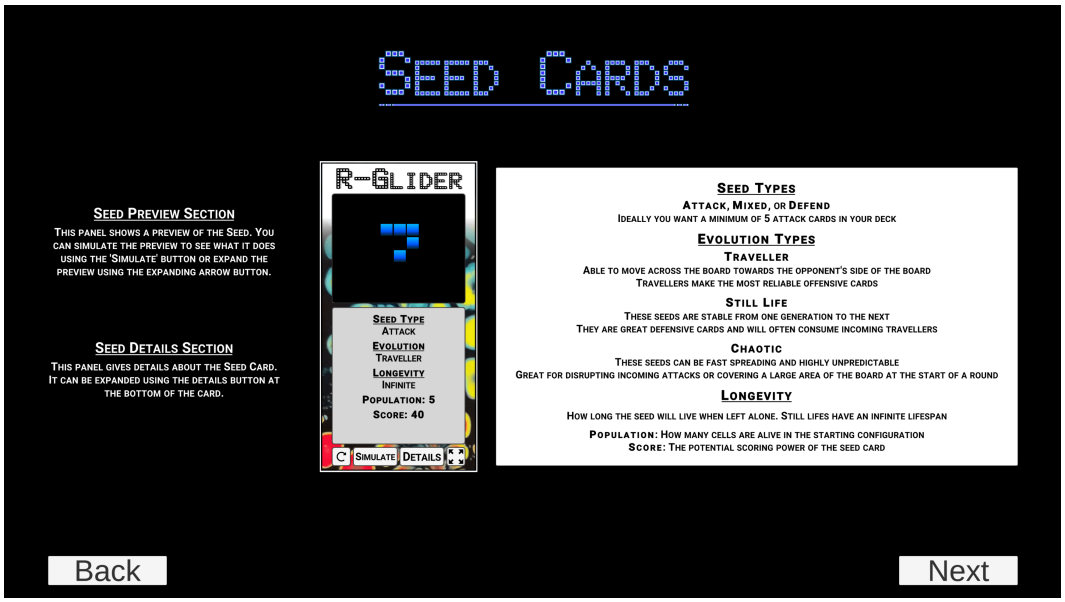

Figure A.3: In-game Tutorial Page Two

Figure A.4: In-game Tutorial Page Three

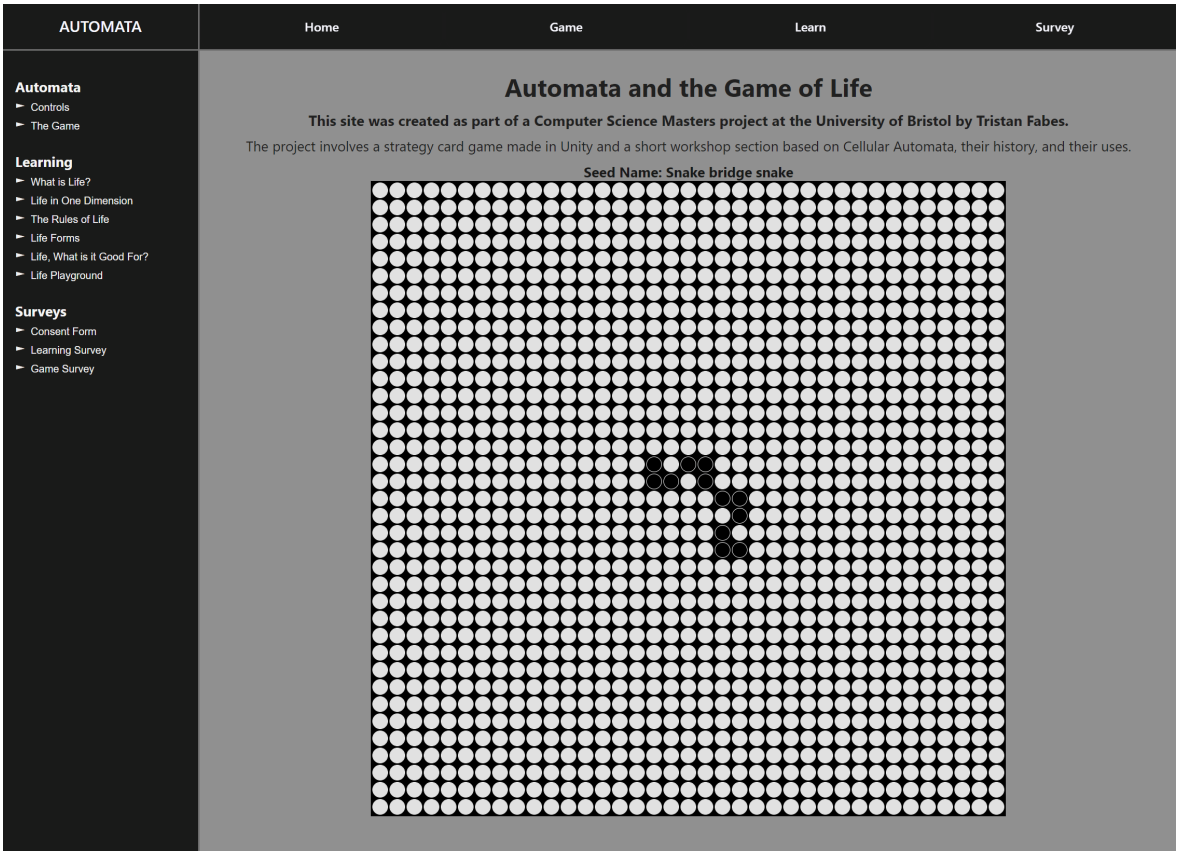

Figure A.5: Web Application Home Page

Bibliography

[1] N. Cohen and L. Squire. Preserved learning and retention of pattern-analyzing skill in amnesia: Dissociation of knowing how and knowing that. *Science*, 210(4466):207–210, 1980.

[2] I. Bogost. Persuasive games: The proceduralist style. *Gamasutra. com,[Online], January*, 21, 2009.

[3] V. Rideout and M. Robb. Media use by kids age zero to eight. *The Common Sense Census*, 2020.

[4] H. O'Neil, R. Wainess, and E. Baker. Classification of learning outcomes: Evidence from the computer games literature. *The Cirriculum Journal*, 16(4):455–474, 2005.

[5] F. Blumberg, K. Deater-Deckard, S. Calvert, R. Flynn, C. Green, D. Arnold, and P. Brooks. Digital games as a context for children's cognitive development: Research recommendations and policy considerations. *Social Policy Report*, 32(1):1–33, 2019.

[6] G. Kacmaz and A. Dubé. Examining pedagogical approaches and types of mathematics knowledge in educational games: A meta-analysis and critical review. *Educational Research Review*, page 100428, 2021.

[7] Plato and A. Bloom. *The Republic, Book VII*. New York: Basic Books, 1968.

[8] T. Malone. Toward a theory of intrinsically motivating instruction. *Cognitive science*, 5 (4):333–369, 1981.

[9] M. Gardner. The fantastic combinations of John Conway's new solitaire game of life. *Sc. Am.*, 223:20–123, 1970.

[10] L. Thomas, M. Ratcliffe, J. Woodbury, and E. Jarman. Learning styles and performance in the introductory programming sequence. *ACM SIGCSE Bulletin*, 34(1):33–37, 2002.

[11] A. Thomas, L. Sherrell, and J. Greer. Using software simulations to teach automata. *Journal of Computing Sciences in Colleges*, 21(5):170–176, 2006.

[12] H. Lilly. The use of cellular automata in the classroom. In *Supercomputing'95: Proceedings of the 1995 ACM/IEEE Conference on Supercomputing*, pages 16–16. IEEE, 1995.

[13] S. Coombes. The geometry and pigmentation of seashells. *Department of Mathematical Sciences, University of Nottingham, Nottingham*, 2009.

[14] M-A. Tsompanas, I-A Fyrigos, V. Ntinas, A. Adamatzky, and G. Sirakoulis. Cellular automata implementation of oregonator simulating light-sensitive belousov–zhabotinsky medium. *Nonlinear Dynamics*, 104(4):4103–4115, 2021.

[15] T. Staubitz, R. Teusner, C. Meinel, and N. Prakash. Cellular automata as basis for programming exercises in a mooc on test driven development. In *2016 IEEE International Conference on Teaching, Assessment, and Learning for Engineering (TALE)*, pages 374–380. IEEE, 2016.

[16] K. Becker. Teaching with games: the minesweeper and asteroids experience. 2001.

[17] J. von Neumann. The general and logical theory of automata. In *Systems Research for Behavioral Sciencesystems Research*, pages 97–107. Routledge, 2017.

[18] J. Schiff. *Cellular automata: a discrete view of the world*. John Wiley & Sons, 2011.

[19] E. Moore. Machine models of self-reproduction. In *Proceedings of symposia in applied mathematics*, volume 14, pages 17–33. American Mathematical Society New York, 1962.

[20] Play game of life, . URL `https://playgameoflife.com/`.

[21] Conway's game of life, . URL `http://www.cuug.ab.ca/dewara/life/life.html`.

[22] Conway's game of life, . URL `https://academo.org/demos/conways-game-of-life/`.

[23] Nathaniel Johnston. Lifewiki. URL `https://conwaylife.com/wiki/Main_Page`.

[24] Chris Lipa. Cornell math explorers' club. URL `http://pi.math.cornell.edu/~lipa/mec/lesson6.html`.

[25] Drew Blaisdell. Life competes. URL `http://lifecompetes.com/`.

[26] Michael Huang. Game of life and death, . URL `https://popclom.github.io/GOLAD/`.

[27] Michael Huang. Game of life and death application, . URL `https://apps.apple.com/us/app/game-of-life-and-death/id1156743291`.

[28] David Brin. *Glory Season*. Hachette UK, 2011.

[29] Ingo Berg. Belto forion. URL `https://beltoforion.de/en/recreational_mathematics/game_of_life.php#idStart`.

[30] Briano Eno. Generative music. Imagination Conference, San Francisco, 1996. URL `https://inmotionmagazine.com/eno1.html`.

[31] Thomas W Malone and Mark R Lepper. Making learning fun: A taxonomy of intrinsic motivations for learning. In *Aptitude, learning, and instruction*, pages 223–254. Routledge, 2021.

[32] Nancy Karweit. Time-on-task reconsidered: Synthesis of research on time and learning. *Educational leadership*, 41(8):32–35, 1984.

[33] M. Griffiths, D. Kuss, and D. King. Video game addiction: Past, present and future. *Current Psychiatry Reviews*, 8(4):308–318, 2012.

[34] M. Lepper, D. Greene, and R. Nisbett. Undermining children's intrinsic interest with extrinsic reward: A test of the" overjustification" hypothesis. *Journal of Personality and social Psychology*, 28(1):129, 1973.

[35] Jean Piaget. *Play, dreams and imitation in childhood*. Routledge, 2013.

[36] M. Csikszentmihalyi. Intrinsic rewards and emergent motivation. *The hidden costs of reward: New perspectives on the psychology of human motivation*, 24(3):205–216, 1978.

[37] D. Berlyne. Conflict, arousal, and curiosity. 1960.

[38] M. Habgood. *The effective integration of digital games and learning content*. PhD thesis, University of Nottingham Nottingham, 2007.

[39] Fiona Art. Abstract painting. URL `https://www.pexels.com/photo/colorful-abstract-painting-3631430/`.

[40] J. Hybek and Endel. Unity websocket webgl. URL `https://github.com/jirihybek/unity-websocket-webgl`.

[41] A. Nyqvist and J. Rutqvist. The impact of colour themes on code readability, 2019.

[42] Uplers Email. In the limelight: Dark mode in emails. URL `https://email.uplers.com/infographics/dark-mode-in-emails/`.

[43] DevSoft Baltic. Surveyjs. URL `https://surveyjs.io/`.

[44] Francesco Berto and Jacopo Tagliabue. Cellular automata. 2012.

[45] Andrew Ilachinski. *Cellular automata: a discrete universe*. World Scientific Publishing Company, 2001.

[46] Reza Zadeh. Monte carlo search tree. URL `https://stanford.edu/~rezab/classes/cme323/S15/projects/montecarlo_search_tree_presentation.pdf`.

[47] G. Faraco, P. Pantano, and R. Servidio. The use of cellular automata in the learning of emergence. *Computers & Education*, 47(3):280–297, 2006. ISSN 0360-1315. doi: https://doi.org/10.1016/j.compedu.2004.10.005. URL `https://www.sciencedirect.com/science/article/pii/S0360131504001526`.