Digitizing Machi Koro

Carleton University

COMP4905 – Honours Project

Written By: Nick Martino, 100934881

Supervisor: Dr. Jean-Pierre Corriveau

December 15, 2017
Abstract

The goal of this project was to create a digital, fully networked and tested, version of the board game Machi Koro. This project was originally undertaken to sharpen my programming skills in a few areas of interest, specifically automated testing, software design, and online networking. To this end all three goals have been a success. Through the size and scope of work required, building the project in a TDD (Test Driven Development) way has proven beneficial several times and the software design and patterns used helped me to keep the code base easy to work with even as it scaled ever larger. The online learning goals have been a significant success, as the program not only works locally, but cross platform over the internet.

This project was a large success, and achieved all the goals I aimed for at the start.
Acknowledgements

I would like to thank the developers of Machi Koro, Masao Suganuma and Noboru Hotta, the company that published it originally, Grounding Inc, and the North American publishers, IDW Games and Pandasaurus Games.

I would also like to thank my supervisor, Dr. Corriveau.

Finally, I would like to thank the developers of Unity, and the contributors at StackOverflow. This project would have been significantly harder had either one of those not been created first.
Contents

Abstract........................................................................................................................................2
Acknowledgements....................................................................................................................3
Introduction ..................................................................................................................................5
Motivation....................................................................................................................................5
Requirements.................................................................................................................................5
Rules of the Game........................................................................................................................9
  Edge Case Rules .........................................................................................................................11
Playing the Game........................................................................................................................12
Methodology..................................................................................................................................14
  Why Unity?...................................................................................................................................16
Software Strategies.......................................................................................................................17
  User Stories................................................................................................................................17
Automated Testing ........................................................................................................................19
  PlayerTests ..................................................................................................................................19
  VictoryCardTests .......................................................................................................................20
  CardTests ....................................................................................................................................20
  CafeTests and FRTests.................................................................................................................21
  ShoppingMallTests.....................................................................................................................22
  PurpleTests..................................................................................................................................22
  AI Tests .......................................................................................................................................23
Online Play......................................................................................................................................23
Difficult Parts.................................................................................................................................24
  Visually Rolling the Die...............................................................................................................24
  Making the Red Cards ................................................................................................................25
  Making the Purple Cards............................................................................................................25
  Making the Victory Cards ...........................................................................................................26
  Building the AI ...........................................................................................................................26
Results.............................................................................................................................................28
Introduction

The goal of my honours project was to digitize the board game Machi Koro. Specific requirements will be laid out under the ‘Requirements’ section, but the broad strokes included using software engineering strategies and patterns, developing a comprehensive testing suite for the program, and allowing multiple players to play on multiple computers.

Motivation

The original motivation for this project came to me when I moved to British Columbia (from Ontario) for an 8-month co-op term. One of the first things that made me miss home was playing board games with friends. When looking for online board games to play with people, I found that very few of my favorites had a digital version available. This prompted me to start wondering if I could make my own online version of one of them.

When I started to think about my honours project, I thought back to work that I had done in a previous course of Dr. Corriveau’s (COMP3004), making a digital version of a card game with a group. Making a digital version of a more complicated board game on my own seemed like a solid step for my academic goals, that happened to coincide with a personal goal.

Requirements

After emailing back and forth with Dr. Corriveau, the basic outline of the project began to take form. The key requirements as stipulated in the proposal were as follows:
1. The ability to play a game to completion, both vs up to 3 simple AI, and against up to 3 player opponents on LAN.

2. The 15 purchasable building cards and 4 victory cards that are included in the base game. These must function exactly as they would in the board game version.

3. The detailed rules of the game, such as the order that money is received and stolen. As these corner cases make a significant impact on play, these must be in the digital version.

4. User stories for the above 3 main tasks, any subtasks associated with them, and any additional features that are later implemented.

5. Technical documentation detailing the interior function and design.

6. A testing suite, that at a minimum provides a test for every building card, victory card and play state change (ex. Buying a building, winning the game, etc). The testing suite will also contain tests for any additional features that are later implemented, and may contain more tests than the bare minimum.

These requirements were all met and surpassed:

1. In addition to being able to play against up to 3 simple AI, you can play against one of 6 different AI’s. Also, you can play against up to 3 player opponents online or with hot-seat; and you can mix and match AI and human players.

2. All 15 building cards and all 4 victory cards are implemented exactly as they would in the base game.

3. All the detailed corner cases are both accounted for and automatically tested, both with unit and scenario tests. There are several hundred tests and over 7000 lines of code just for testing.

4. User stories are detailed below.
5. Technical documentation is detailed below

6. The testing suite covers every case, both the cards and actions and values that can and should interact with each other, and the cards, actions and values that do not interact with each other (to make sure that they don’t). This includes both unit and scenario tests.

Additionally, several potential features were outlined that would be considered extra, not necessary, but interesting to implement. Of these, the following were completed:

- Multiple types of AI, with a focus on different strategies.
- General polish of visuals and gameplay.
- Ability to play against players online
- Ability to play cross platform
- Ability to play against a mix of AI and players.

There are 6 unique AI strategies, each of which plays the game considering cards that it has, that other players have, the coins it has on any given turn (sometimes it chooses to hold back to save up, sometimes it makes a less optimal move on one turn to increase its coins for the following turn), and probability of certain numbers being rolled before it next gets to act.

The game is generally at a high level of polish. One standout is the way the UI (User Interface) responds to the game state; such as fading out cards that the currently active player cannot afford, or highlighting cards that were triggered this turn. Another polish example is the fully 3D rolling die, which rolls as you drag it across the table and spins until it lands on a number. This was originally created to help mimic the
tactility of the board game’s physical pieces, and it succeeds. In addition, you can move the various
cards around the table to accommodate your personal preferences better.

You can face other players from anywhere, to anywhere. The server code is stored on my personal
website, and runs from there.

Cross platform works because the web architecture is platform independent; in addition, it can be
played with a touch screen.

Playing against a mix of AI and players is self-explanatory; but you can also play against this mix while
playing over the internet and cross platform. For example, you can have one player playing on a tablet,
one on a PC, and 2 AI in the game, and it all fits together.

Overall, not only have the initial requirements been met and surpassed, but many of the additional
features have made it into the final product.
Machi Koro is a turn based strategy game, where you develop a town by purchasing buildings. Your eventual goal is to purchase all 4 victory buildings, at which point you win the game.

Each turn, the following actions occur:

1) The current player rolls the die.

2) Each player looks at their buildings, and checks which ones activate based on the number rolled and whose turn it is. Buildings typically generate coins.

3) The current player can then spend coins to purchase up to one building.

4) Check to see if anyone has all 4 victory buildings

5) If not, pass the turn to the next player
6) If so, end the game, that player wins.

The game is made interesting by the unique buildings you can purchase. Each of these had to be implemented in the digital version of the game.

**General Cards**

- **Wheat Field** – **Cost 1**, activates on 1 on Any Turn, generates 1 Coin, Counts as Farm
- **Ranch** – **Cost 1**, activates on 2 on Any Turn, generates 1 Coin, Counts as Cattle
- **Forest** – **Cost 3**, activates on 5 on Any Turn, generates 1 Coin, Counts as Production
- **Mine** – **Cost 6**, activates on 9 on Any Turn, generates 5 Coins, Counts as Production
- **Apple Orchard** – **Cost 3**, activates on 10 on Any Turn, generates 3 Coins, Counts as Farm
- **Bakery** – **Cost 1**, activates on 2-3 on Your Turn, generates 1 Coin, Counts as Cooking
- **Convenience Store** – **Cost 2**, activates on 4 on Your Turn, generates 3 Coins, Counts as Cooking
- **Cheese Factory** – **Cost 5**, activates on 7 on Your Turn, generates 3 Coins per Cattle
- **Furniture Factory** – **Cost 3**, activates on 8 on Your Turn, generates 3 Coins per Production
- **Fruit and Vegetable Market** – **Cost 2**, activates on 11-12 on Your Turn, generates 2 Coins per Farm
- **Cafe** – **Cost 2**, activates on 3 on Anyone Else’s Turn, steals 1 Coin from Current Player, Counts as Coffee
- **Family Restaurant** – **Cost 3**, activates on 9-10 on Anyone Else’s Turn, steals 2 Coin from Current Player, Counts as Coffee
- **Stadium** – **Cost 6**, activates on 6 on Your Turn, steals 2 Coins from Everyone
- **TV Station** – **Cost 7**, activates on 6 on Your Turn, steals 5 Coins from Your Choice
- **Business Center** – **Cost 8**, activates on 6 on Your Turn, Swaps Two cards (One you control, one you do not).
Victory Cards

- **Train Station – Cost 4**, You may roll 1 or 2 dice.
- **Shopping Mall – Cost 10**, Each of your Cooking and Coffee buildings earn +1 coin
- **Amusement Park – Cost 16**, If you roll doubles, take another turn after this one
- **Radio Tower – Cost 22**, Once every turn, you can choose to re-roll your dice.

Each player starts the game with 3 coins, a Wheat Field and a Bakery.

While some of these (the ones that only affect you, and can only happen on your turn) were trivial to build, others, such as the Business Center, were extremely difficult, especially in online play. In addition, several of the cards (Business Center, Train Station, Radio Tower) required additional UI elements to be built, and some affected the general flow of the gameplay (such as Amusement Park and Radio Tower).

**Edge Case Rules**

There are a set of rules that are more specific than the ones outlined above, and as they have a major impact on gameplay, they needed to be addressed and implemented too.

These include:

- When you reroll with a Radio Tower, you must use the same number of die as the first time.
- You cannot purchase more than 1 of each ‘Business Card’ (Stadium, TV Station, Business Center)
- You cannot swap a Stadium, TV Stadium or Business Center using the Business Center
• When you roll a 3, 9 or 10, and have coins taken from you, you receive your coins after the coins are removed; you cannot go into debt. This ensures that in the early game, any 3’s your roll will always leave you with at least one coin.

• When you are giving other players’ coins, it goes indie reverse turn order. If you owe 3 people 2 coins each, and you only have 4 coins, you give the two people to your right 2 coins each.

• You must roll before you can buy

• There are 6 of each card available except Wheat Field and Bakery (Which can have between 7 and 10 depending on the number of players, 6 in the buy section and each player starts with 1 of each), and the Business Cards of which there are only 4 of each available.

Playing the Game

This project attempts to mimic the board game where possible; and as such playing the digital version is much like playing the physical one in many respects. There was a focus on making a strong and easy to
use UI, and so much of playing the game is very self-explanatory. However, in addition to that, below is all the details on how to play.

To start a game, launch the program and select one of the three options on the first screen.

- ‘Face AI’ is to face computer opponents
  - Once selected, click on the number of AI you want to face.
- ‘Hotseat’ is to play against other players on the same computer
  - Once selected, click on the number of other players you want to play with.
- ‘Online Play’ is to face other players over the net
  - Once selected, you are loaded into the lobby. Other players launch the game and click online play to join the lobby.
  - Any player can start the game by selecting one of the AI options at the bottom (Number of AI to include in the game).
    - The game will prevent you from including more AI than can fit in 4 players

Once loaded into the game, the first player is up. A small orange line will be by the player whose turn it is. To end your turn without buying anything, after you roll, click on this line.

To roll the dice, click and drag it, then let go. After rolling, the cards you can afford will light up in the buy section, and the cards that triggered will light up in each players’ boards. Click on one of the cards in the buy section to buy it. There are a limited number of each card available and remaining, which is visible when you mouse over a card in this section.

Once you select a card, your turn will end automatically, and the next player will go. If it is an AI, they will automatically start to play. To toggle AI delay on actions on or off, press spacebar on computer.
Prompts will show up when you have either the Train Station or the Radio Tower that allow you to input how many dice you want to roll, and ask if you want to reroll your die respectively.

When you activate the Business Center, you click on one card from amongst yours, and then one from your opponents, and the two will exchange control (they will also visually swap on the board).

Methodology

Building a digital version of a board game is a very visual task; and the tools I chose to use to work with reflect that. My primary tool of choice was Unity, using C# as a scripting language. Unity helps build the visual elements, and has significant available help online, as it is a widely used engine. When it came to the server, I used PHP, primarily due to familiarity, alongside MySQL for backend data storage.

In Unity, programs are built around ‘Scenes’ which contain visual elements and objects that all run simultaneously. Machi Koro contains 3 Scenes.

- Intro
  - This is the initial screen the players load into. It loads up basic game data, and presents interactive menus for players to select what mode they want to play and how many other players/AI they want to play with (In the case of VS AI and Hotseat play).
  - As Unity is particularly well suited for them, the Intro Scene uses a Singleton object, MenuManager, to handle updating the visuals and taking in user input.
- Each button is, however, its own object, and knows what it is. Manu Manager is used to take in the input after it is complete and load the game appropriately.

- Lobby

  - This is the screen you are taken to when selecting Online Play.
  - Much like the Intro Scene, Lobby is managed by a Singleton LobbyManager, but in this case, it is a much more significant class.
  - Lobby Manager deals with connecting to the server, determining who is in the lobby, and finally starting the game once a player has selected how many AI they want.

- Main

  - Main is the largest Scene, and contains the bulk of the programming.
  - Here the Singleton pattern is used twice times.
  - Game Manager handles all the game logic that needs to flow through different objects, such as handling turns, holding the players, and storing all the data used in the game.
  - Online Manager is used only in online mode, as the point of contact between the game (represented by Game Manager) and the backend server.
  - Other key classes include the CardFactory, which uses the Factory pattern to build cards as the players need them, and the Player Controller, which uses a modified version of the Strategy pattern for dealing with the AI strategies.
Working with these Scenes for online play are standalone PHP files, which are stored on nmartino.com, and are simply called inside the code with the appropriate POST data. These files interface directly with the SQL table that acts as backend data storage for the game, with each table representing a single game.

**Why Unity?**

Unity was used for this project for several reasons. The single largest of those is that it is a specialized game engine, as opposed to a more general language like Java. In the past, when I have gone to make games with UI elements using general languages and tools, I have ended up rewriting significant parts of the graphics libraries to support the refresh rate of a game. There was no such problem with this in Unity, as the whole thing is created to build games in.

Unity does have several other large advantages. The online support, forums, and docs for Unity development are extensive, and answer many questions that I had asked directly, including version differences in Unity. Another such advantage is the ease of cross-platform development. While some code still needs to change based on platform, it is primarily how the user interacts with the game, rather than the backend logic itself. Unity handles all the porting logic behind the scenes. Rounding out the reasons I chose Unity are that I find it easy and comfortable to work in; everything is where I expect it to be, and C# is my most liked programming language. Unity-C# combines many of the things I like in a language; strongly typed, automatic garbage collection, object oriented, etc.

On top of these general reasons for picking Unity, it also greatly simplifies many goals I had set out. Unity has built in automated testing; no need to configure anything like JUnit. Making networks calls is simple and doesn’t require an external library, it’s also built in. The UI can be visually built on the
The combination of these technical advantages and the general advantages above are why I selected Unity to work in for this project.

Software Strategies

As mentioned above, a few specific software patterns were heavily used in this project. The first of these was the Singleton. Singletons excel in Unity because the data is so naturally spread out by the Scene structure, that some way to group critical parts back together is typically needed.

Another excellent pattern for Unity development is the Factory pattern. Since the logic of a program is based on a large set of objects, being able to quickly and easily create new ones of a needed type is very useful.

The final specifically used pattern was a modified version of the Strategy pattern. As interfaces and class extension can get tricky in Unity, instead a single function is used to automatically swap between the appropriate function based on a variable ‘aiStrat’. This functions nearly the same as the pattern, only fits more easily into the Unity and C# package.

One final thing to note about the Player and PlayerController class. The class is split such that PlayerCotroller handles all the game logic, and Player handles all the UI elements. It was split this way to enable simpler automated testing.

User Stories

A series of User Stories were developed before work began, and several more after work was underway, to ensure that all required functionality was implemented. While writing User Stories is less critical in a
single person project, it still helped as a sort of checklist, and a way to organize early thoughts about the project.

- As a player, I want to be able to roll the dice on my turn
- As a player, I want to be able to purchase a card after I roll the die on my turn
- As a player, I want to be able to pass the turn without buying a card
- As a player, I want to be able to receive coins from blue cards on my turn and other players’ turns
- As a player, I want to be able to receive coins from green cards on my turn
- As a player, I want to be able to take coins from others using purple cards on my turn
- As a player, I want to be able to swap cards with the Business Center card
- As a player, I want to be able to get coins from other players on their turns using red cards
- As a player, I want to be able to roll 1 or 2 dice once I have the train station
- As a player, I want to be able to get extra coins from the Shopping Center
- As a player, I want to be able to take an extra turn with the Amusement Park
- As a player, I want to be able to reroll a die if I choose with the Radio Tower
- As a player, I want to be able to win the game if I get all 4 victory cards
- As a user, I want to be able to select the number of AI I face
- As a user, I want to be able to face other players in hot-seat
- As a user, I want to be able to face other players online
- As a user, I want to be able to face both players and AI at the same time
- As a user, I want to be able to watch the AI play out their turns so that I can follow the game
- As a user, I want to be able to skip AI action delay so that I can speed up games
As a user, I want to be able to move cards around the board, so that it feels more like a physical table.

I used player to reference things that were done in game, and user to reference UI and options. This helped me keep those things separate in my thoughts, and eventually the code.

### Automated Testing

To fully and completely test this game, several hundred tests were made, both scenario and unit tests, totaling over 7000 lines of test code. The tests are broken down into 8 different test classes.

![Test Runner](image.png)

- **PlayerTests**

  The PlayerTests are the most basic tests; what happens when the player takes basic actions. These include things such as:

  - Can they receive different numbers of coins?
  - Can they gain cards?
  - Can they end their turn?
  - Can they roll the die?
  - And more along the same lines.
These tests, despite being simple, are the ones that fail most often when making changes. This was somewhat unexpected, and was one of the things that taught me the most during this project. If I had not written these simple tests, several bugs would have gone unnoticed for much longer.

**VictoryCardTests**

VictoryCardTests is the next set, and it is the smallest, containing only 4 scenario tests.

- Test the Amusement Park twice (2 player and 3 player)
- Test the Train Station twice (Rolling one die, and rolling two die)

These tests were very useful when developing, as it helped to flesh out the order that actions would be occurring when these Victory Cards were in use.

**CardTests**

CardTests contains the unit and scenario tests for all the buyable cards except those tested by VictoryCardTests, ShoppingMallTests, FRTests, CaféTests, and PurpleTests. Initially CardTests contained all of those too, but each of those categories is significant enough that it caused the class to become bloated, so they were separated. The tests for each card vary based on type.

- **Blue Cards**
  - Simple unit tests to make sure that they work on your turn, and on another players’ turn
- **Bakery and Convenience Store (Simple Green Cards)**
  - Simple unit test to make sure that they work on your turn, and not on another players’ turn
- **Other Green Cards**
Simple unit tests like the simple green cards

Longer scenario tests based on what is on the board at the time, and shifting board states.

Much like the VictoryCardTests, these tests were primarily useful in development, and rarely broke after the work was initially done to implement them.

CafeTests and FRTests

These two sets of tests are grouped together because they are very similar. They test the red cards, which trigger on opponents turns, and take coins from other players. These were some of the most complicated cards to implement in the game, and were correspondingly difficult to test. Even the simplest of tests in these classes are scenario tests, and these two tests (coupled with the ShoppingMallTests below) make up nearly half of the testing in the program.

- Test the trigger on your turn
- Test the trigger on your opponents’ turn
- Test when one person has coins, both have coins, the other has coins
- Test with multiple Café’s and Family Restaurants.
- Test with only able to pay out partial debt
- Test the ordering of paying out partial debt
- And more.

The difficulty in implementing these cards and tests was unexpected, to say the least.
ShoppingMallTests

ShoppingMallTests were all just testing the Shopping Mall victory card. Much of the expansiveness of these tests comes from the need to retest the Café and Family Restaurant with the Shopping Mall in play. These tests were set up once, and then have run ever since.

PurpleTests

PurpleTests are scenario tests that test the purple cards in the game (Stadium, TV Station, and Business Center). These, along with the Victory Cards, are the most complicated cards in the game. Each one is not only different from each other, but different from any other actions in the game.

- Stadium
  - The most straightforward of the purple cards. Checks that you get coins on your turn, that you don’t overtake from people, and that you don’t get coins on other’s turns.

- TV Station
  - Tricky to test because it has an element of choice that needed to be decoupled from the UI.
  - Tests that you take from people only on your turn, don’t overtake, and take from only the person selected.

- Business Center
  - Very complicated to build and test. It requires a UI element, and interaction with other players’ cards (something that no other effect in the game deals with).
Tests that it only triggers on your turn, that it swaps cards correctly, and that it swaps cards even when you decide the same card for both players.

While these tests were much simpler than the implementation of these cards, they still required navigating around the UI, and as such were difficult to construct. They have, however, caught several bugs after implementation and have been well worth the difficulties in creating them.

AI Tests
AI tests are scenario tests that check to make sure the AI is functioning correctly. These were simple to set up and no real issues have been found with them since.

Online Play
Building online capabilities was the most challenging part of the project, as it is an area in which I am not well-versed. I have, however, done significant work with databases as part of a few co-op terms. This led to a database focused server.

When you select Online Play and open the lobby, it gets who is currently in the lobby from the database, then adds you to that group, and reuploads. It then loads in the lobby. Whenever any player starts the game, it sends a start signal to all other players and appends START:# (# is the number of AI) to the field in the table. This is the cue that the game should start, and the lobby is closing.

Once inside the game the board state is serialized and sent to the database every time a player ends a turn. This signals to all the others that they should update, including who the next player is. This keeps the board state in sync through trickier operations like the red and purple cards, especially the Business Center.
By doing it this way, all the game logic functions exactly as it would in single player, which keeps it very consistent.

**Difficult Parts**

Broadly speaking, there were 6 sticking points in the project. These were:

- Visually rolling the die
- Making the red cards
- Making the purple cards
- Making the victory cards
- Building the AI
- Online Play

**Visually Rolling the Die**

While this was not a critical part of the project, it was implemented to make the game feel more like the board game version. There were several tricky parts to it. First was the implementation of a 3D object in a 2D game. There were many problems with it clipping behind the board, and then being unclickable. This was solved by physically moving its location forward, so that it is far in front.

Another major issue was reading the side that is facing up. In the end, instead of reading it, the die simply rolls and lands on the side that the random number generator in the backend pulled up as the result.
Making the Red Cards

The red cards were built after the blue and green, both of which were straightforward. However, this meant that the framework was already built for card activation. This was a significant problem when it came to the case where other players did not have enough coins to fully pay out. Under the already built framework, the player whose turn it was received coins; there was no hook for other effects to occur and return values (such as how many coins the player could pay out). Adding that, and all the checks for the various corner cases (see CafeTests and FRTests for examples) was a large amount of work.

Making the Purple Cards

Each of the purple cards represented its own challenge. The Stadium was built using most of the same technology as the red cards, and so after those were done it was not too difficult. Both the TV Station and the Business Center, however, required additional UI elements and points of interaction after a player rolls, but before they buy a card. In addition, the Business Center required another players’ board to be updated during the wrong turn. Each of these issues required extensive work to the base code, and created some of the messiest parts of the program.
Making the Victory Cards

Like the purple cards, every victory card is unique; none of them follow a template like the blue, green or red cards. The Amusement Park was surprisingly simple to implement, as when it triggered it just set a boolean and when the turn should be passed, it hard-set the next player to the same as current player. By contrast, the Train Station and Radio Tower were very difficult. By the time I was working on these, I had figured out that my program didn’t do well with things that altered the basic gameplay flow, such as prompts before or right after the die roll respectively. While testing these was not too difficult, implementing them in the first place required the last major code-rework of the project. Once again in contrast, the Shopping Mall was simple to implement but required extensive testing. This is because it affects the red cards, already the largest set of tests, and doubles up on all the red tests as well as more of its own.

Building the AI

The basic loop of the AI is straightforward. It reads the board state and probability of die rolls, alongside a strategy it generates at the start of the game, and tabulates these values together into a heuristic. This is then compared against the cards it can afford, and that are still available, and it makes a purchase based on that. The difficulty comes from other decisions;

- What to swap with a Business Center
  - The AI will value each card it controls, and each card all other players control, and swap the lowest from it, and the highest from an opponent.
  - While this does not consider how much a choice hurts an opponents’ strategy, it is sufficient to function.
- Purchasing a Business Center is almost never the correct choice, so this case rarely comes up.

- **Who to take from with a TV Station**
  - The AI will count a total coin value of each player (buildings value + 2*victory cards value + coins on hand), and then select the player with the highest coin value and who has at least 4 coins on hand.

- **When to roll two dice using Train Station**
  - The AI ignores other players when deciding this, instead opting to focus solely on its own board.
  - This does create an exploitable weakness with certain AI strategies, where buying Mines and Apple Orchards are significantly better against them.

- **When to reroll using Radio Tower**
  - Like the Train Station solution, it focuses solely on its own board and results.
  - Unlike the Train Station solution, however, this does not create an obvious way to exploit the AI

The AI is not perfect by any means. The early game strategies do not save coins enough, so they tend to get stuck at 2 or 3 out of 4 victory cards and stall there, and the late game strategies get beaten reliably by Mine and Apple Orchard. Despite this, the AI still accurately simulates playing against other people to a large degree, and you can see the different strategies AI employ play out in a game, which adds to the feeling of playing with different people.
Results

This project has been an overwhelming success. In addition to the requirements that were laid out at the start of the term, many extra objectives were met. I set out to improve my TDD, software engineering, and networking skills, and on all three of those fronts I have learned a significant amount. I also personally set out to create an online version of Machi Koro to play with friends as we all go our separate ways following university; and in that regard, it was another success. I am exceedingly happy with the results of this project.