COMP90055 Computing Project

Type: Software Development Project

An Agent-Human Interactive Deception Game Based on Unity

Name:	Ruilin Liu
Student ID:	871076
Supervisor:	Michael Kirley & Peta Masters
Credit Points:	25

Semester 1, 2019

Abstract

Is there a possibility that computers can deceive people with special tricks like magic? In this project, we built a framework of a deception game as a stage for "magic tricks". We provided APIs to support agent development in the future. In this report, we will introduce the deception game, analyze the structure of the system then describe the development process. We will also briefly introduce the API documentation, explain two simple AI strategies and test the functions of the project.

Table of Contents

1. Introduction

1.1 Background

Magic deceives audience through special tricks and props. There might be a possibility that computers can do the same thing. Deceptive AI is an ARC Grant DP180101215 'A Computational Theory of Strategic Deception' Project which aims to deceive people by building stage conjurors techniques into AI agents (“Deceptive AI,” n.d.). In this project, we built a framework of a deception game as a stage for “magic tricks”. We provided APIs to support agent development with different strategies, which will be used to understand whether and how artificial intelligence might deceive people. The deceptive levels of these agents could be tested by playing this game against human players.

1.2 Game Overview

The game is turn-based. There are 2 players, an AI agent and a human player. The objective of AI is to make a chain between any two anchors by depositing red counters, and the human’s objective is to block the chain. On AI’s turn, it controls the shuttle(s) to collect counters from the counter generator(s), then deposits the counters on the board. On human’s turn, players could block one grid location and AI would not be able to deposit counters there anymore.

1.3 Game Engine - Unity

A user-friendly game engine could be helpful for game development. Unity is a powerful game engine across multiple major platforms (“Unity - Multiplatform,” 2019). It supports both 2D and 3D (“Unity - Manual: 2D or 3D projects,” 2019). Moreover, Unity provides complete documentation (“Unity - Manual: Unity User Manual,” 2019) and there are a large number of tutorials online (“Unity Learn,” 2019). Therefore, we use Unity in this project.

1.4 Game Platform - WebGL

Unity allows users to publish projects as JavaScript programs and run in a web browser. As an important component of this project is human-computer interaction, WebGL becomes a convenient choice. It will help to run experiments and collect data easily. However, due to the compatibility limitations of different browsers, the performance of the game varies (“Unity - Manual: WebGL,” 2019). We strongly recommend using Mozilla Firefox or Google Chrome to run this game. Using Apple Safari may cause game stuck while uploading log files to the server at the end of each game.

2. Game Description

2.1 Key Requirements

The key requirements include:

1. Generate a customisable game board.
2. Generate anchors randomly. The minimal distance apart, the number of anchors and the positions should be customisable.
3. Generate four counter generators. The number of counters in each generator should be customisable.
4. The colors of counters in the generators are random by default. Provide an API to control the color proportion.

5. Counters are double-sided. One side is either red, yellow or blue, and another side is white.
6. The number of shuttles should be customisable.
7. The Shuttles can collect counters from generators or from the board. The counters on the shuttles could be turned over programmatically.
8. The Shuttles can deposit counters. The counters are face-down after deposit. The color will reveal a few seconds when an empty shuttle passes over.
9. A timer that displays the remaining time.
10. A log file stores the game processes and results.

2.2 Menu Scene

The project includes two scenes, a menu scene and a game scene. The menu scene is divided into three parts: a main menu, an options panel and a help panel. Click the Play button can start a new game with the default setting.

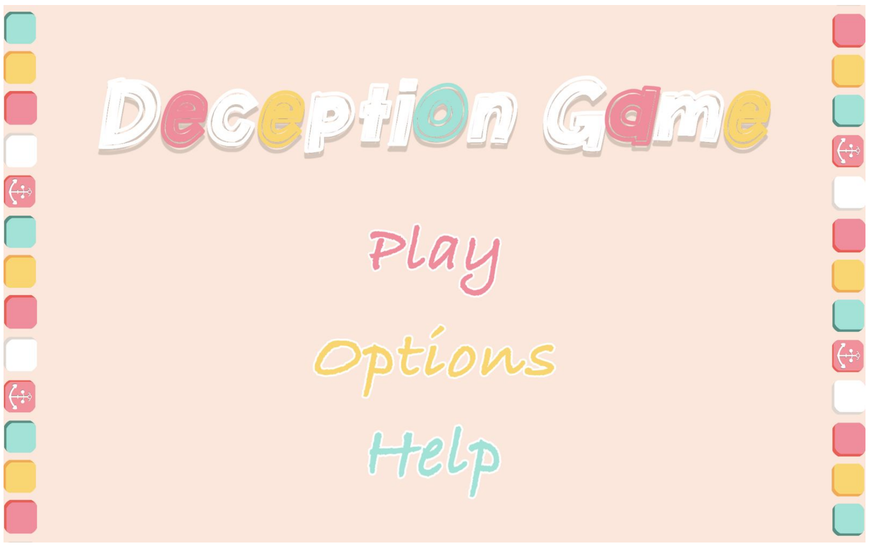


Figure 1. The Main Menu Screenshot

Click the "Options" button will show the options panel. It provides several simple options for the game.

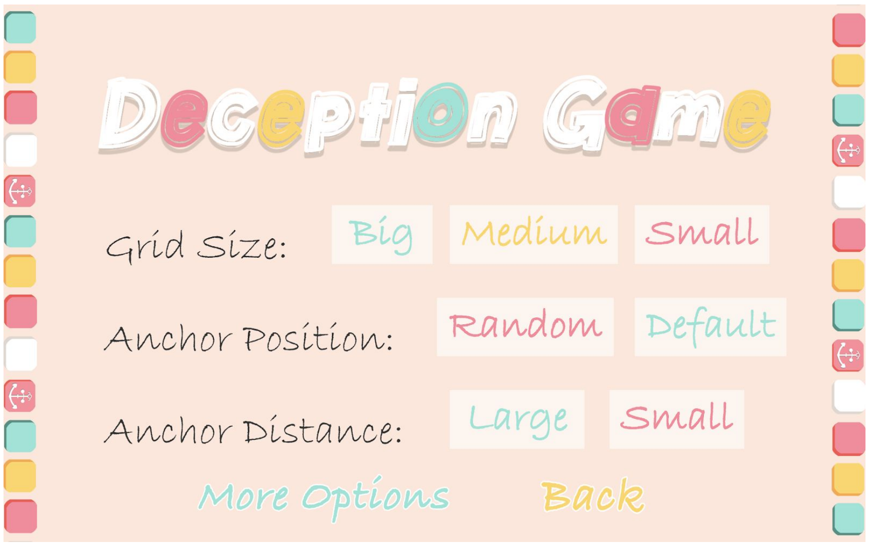
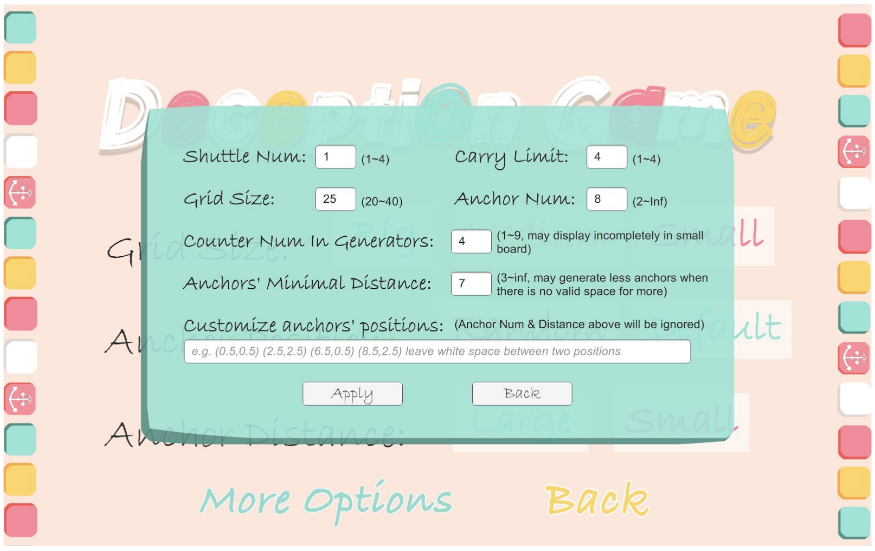


Figure 2. The Options Panel Screenshot

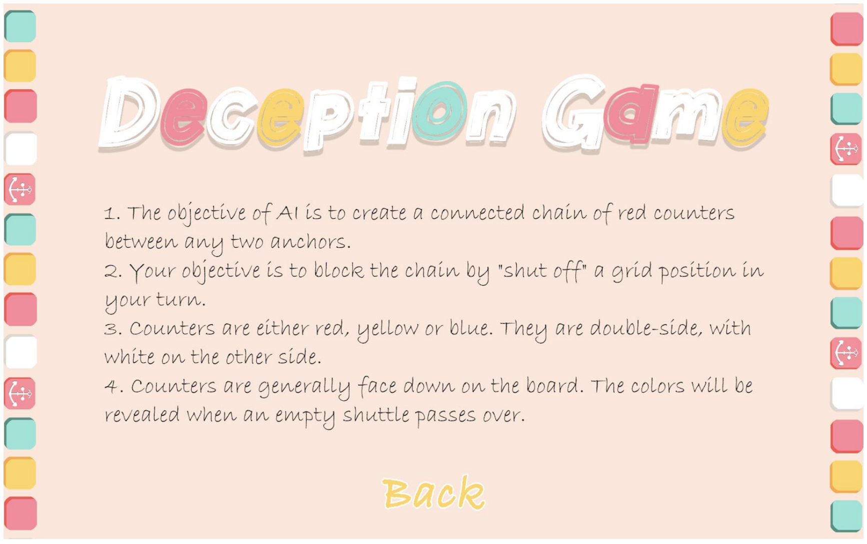
For advanced development needs, more detailed options are listed in “More Options” panel. Note that inappropriate values may cause an incomplete display in some cases. The number of anchors and the minimum distance between the anchors could set to infinity, while it will be ignored when there is no legal position on the board. Users may specify the coordinates of the anchors and the parameters above would be omitted.



Shuttle Num: 1 (1~4)
Carry Limit: 4 (1~4)
Grid Size: 25 (20~40)
Anchor Num: 8 (2~Inf)
Counter Num In Generators: 4 (1~9, may display incompletely in small board)
Anchors' Minimal Distance: 7 (3~inf, may generate less anchors when there is no valid space for more)
Customize anchors' positions: (Anchor Num & Distance above will be ignored)
e.g. (0.5,0.5) (2.5,2.5) (6.5,0.5) (8.5,2.5) leave white space between two positions

Figure 3. The More Options Panel Screenshot

The “Help” panel shows the rules of the game briefly.



1. The objective of AI is to create a connected chain of red counters between any two anchors.
2. Your objective is to block the chain by "shut off" a grid position in your turn.
3. Counters are either red, yellow or blue. They are double-side, with white on the other side.
4. Counters are generally face down on the board. The colors will be revealed when an empty shuttle passes over.

Figure 4. The Help Panel Screenshot

2.3 Game Scene

The game scene holds a timer, a “back to menu” button, a “restart” button, a game board, 4 counter generators, several anchors and shuttles. The screenshot below shows 8 anchors and 3 shuttles. Custom settings could work continuously after “restart”, while they will be abandoned when clicking “back to menu”.

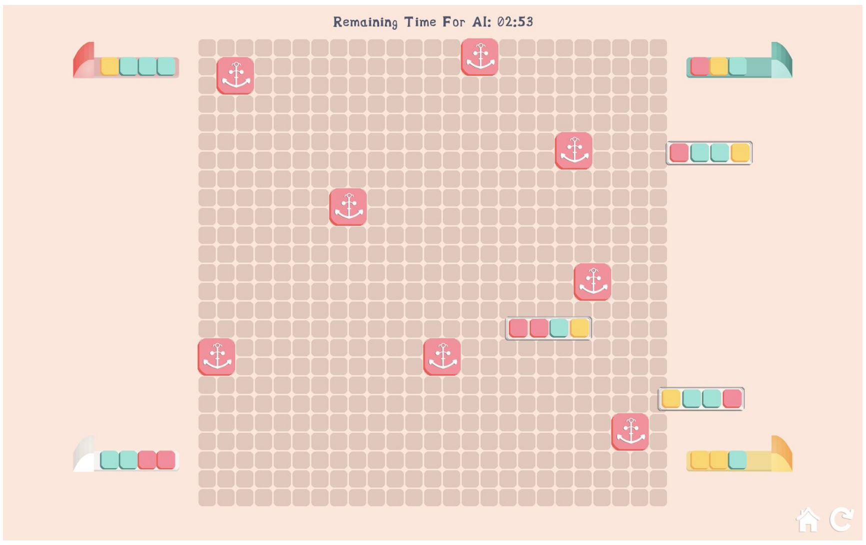


Figure 5. A Multi-Shuttle Game Screenshot

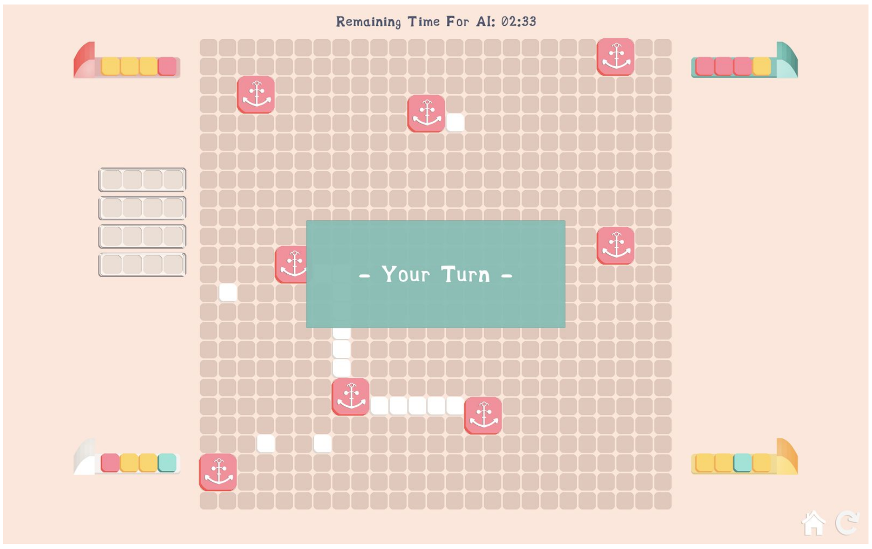


Figure 6. Turn Switch Screenshot

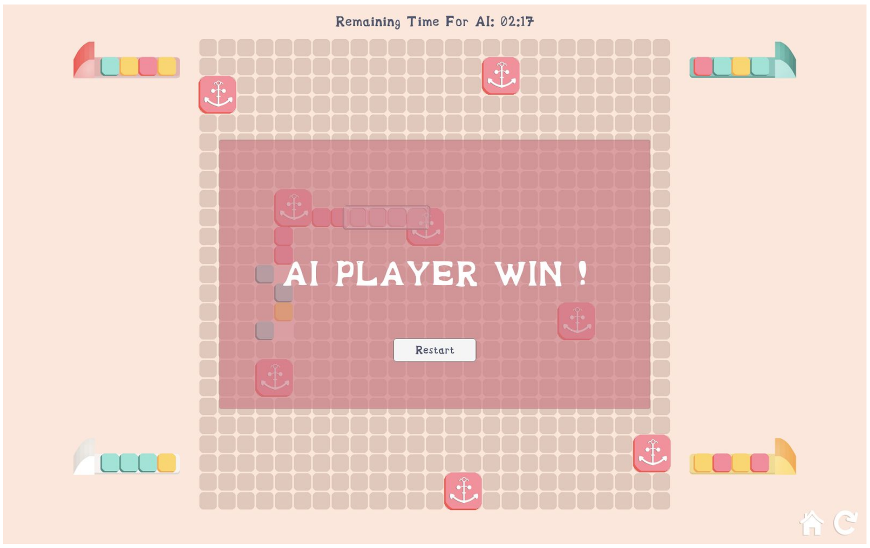
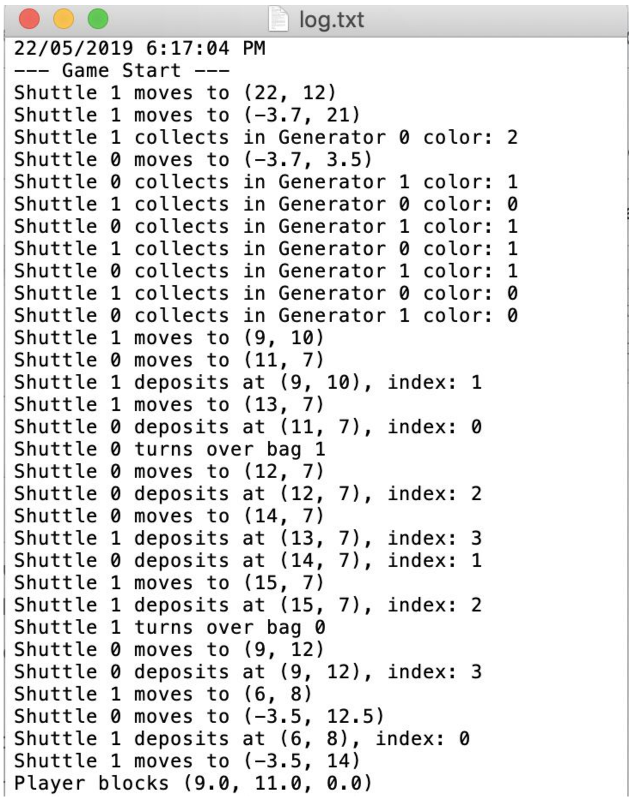


Figure 7. Game Over Screenshot

2.4 Game Log File

At the end of each game, a game log file will be sent to the server. It contains the game time, AI actions, human actions. Sending files to the server may cause the game to stuck for about 10 seconds when using Apple Safari due to the compatibility limitations.

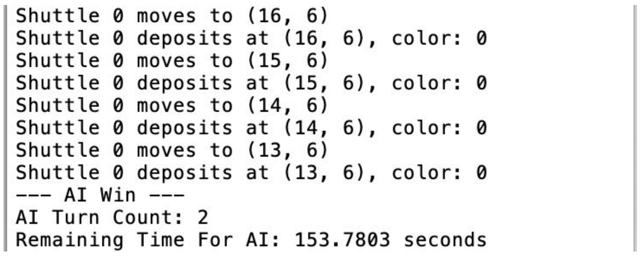
The screenshot below shows the actions of the AI and the human player in one turn. For example, (22, 12) and (-3.7, 21) are coordinates of two counter generators. The shuttles move to the two generators and collect 4 counters respectively. color 0, 1, 2 correspond to red, yellow and blue.



```
22/05/2019 6:17:04 PM
--- Game Start ---
Shuttle 1 moves to (22, 12)
Shuttle 1 moves to (-3.7, 21)
Shuttle 1 collects in Generator 0 color: 2
Shuttle 0 moves to (-3.7, 3.5)
Shuttle 0 collects in Generator 1 color: 1
Shuttle 1 collects in Generator 0 color: 0
Shuttle 0 collects in Generator 1 color: 1
Shuttle 1 collects in Generator 0 color: 1
Shuttle 0 collects in Generator 1 color: 1
Shuttle 1 collects in Generator 0 color: 0
Shuttle 0 collects in Generator 1 color: 0
Shuttle 1 moves to (9, 10)
Shuttle 0 moves to (11, 7)
Shuttle 1 deposits at (9, 10), index: 1
Shuttle 1 moves to (13, 7)
Shuttle 0 deposits at (11, 7), index: 0
Shuttle 0 turns over bag 1
Shuttle 0 moves to (12, 7)
Shuttle 0 deposits at (12, 7), index: 2
Shuttle 0 moves to (14, 7)
Shuttle 1 deposits at (13, 7), index: 3
Shuttle 0 deposits at (14, 7), index: 1
Shuttle 1 moves to (15, 7)
Shuttle 1 deposits at (15, 7), index: 2
Shuttle 1 turns over bag 0
Shuttle 0 moves to (9, 12)
Shuttle 0 deposits at (9, 12), index: 3
Shuttle 1 moves to (6, 8)
Shuttle 0 moves to (-3.5, 12.5)
Shuttle 1 deposits at (6, 8), index: 0
Shuttle 1 moves to (-3.5, 14)
Player blocks (9.0, 11.0, 0.0)
```

Figure 8. Log File Screenshot - 1

At the end, the log records the results of the game and a simple summary.



```
Shuttle 0 moves to (16, 6)
Shuttle 0 deposits at (16, 6), color: 0
Shuttle 0 moves to (15, 6)
Shuttle 0 deposits at (15, 6), color: 0
Shuttle 0 moves to (14, 6)
Shuttle 0 deposits at (14, 6), color: 0
Shuttle 0 moves to (13, 6)
Shuttle 0 deposits at (13, 6), color: 0
--- AI Win ---
AI Turn Count: 2
Remaining Time For AI: 153.7803 seconds
```

Figure 9. Log File Screenshot - 2

3. Project Development

3.1 Agile Methodology

This project is developed under Agile. In each sprint, we implemented some of the requirements of the game. In the process of experiencing the game, we discussed the game balance and deceptive power with the clients. Some requirements were modified after discussion, and we implemented the new features in the next sprint.

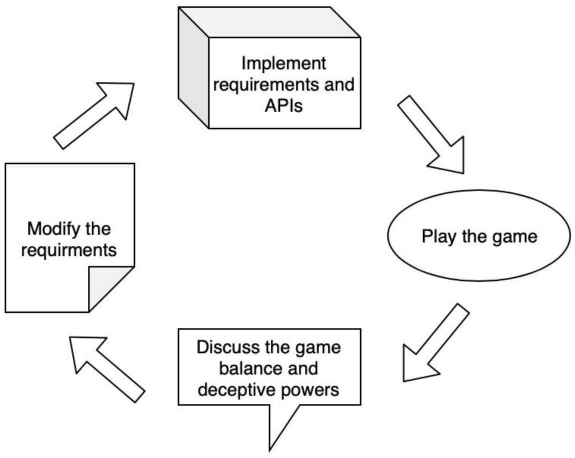

Implement requirements and APIs
Play the game
Discuss the game balance and deceptive powers
Modify the requirments

Figure 10. The Development Process in Each Sprint

The structure of the system kept changing during agile development. Only some of the main components existed in the first version and others are added later. The final structure described below is the result after multiple refactorings.

3.2 Game Components

The main menu connected to the options menu and help menu. The GameParameters class contains all parameters of the game and the OptionMenu class holds methods and buttons to modify the parameters in GameParameters.

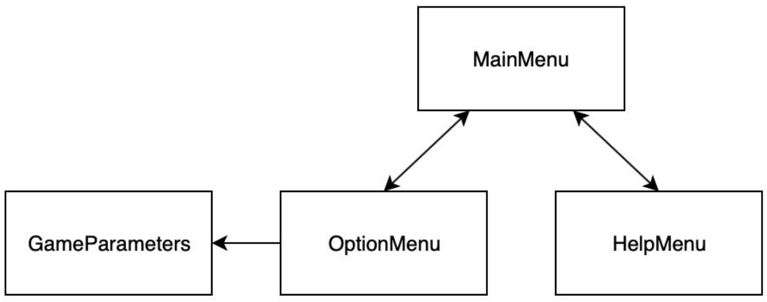


Figure 11. Main Components of the Menu Scene

The GameManager controls the logic of the game. At the beginning of each game, it calls the BoardGenerator to setup the board, such as the generating tiles, generators, shuttles and initializing the game camera based on the board size. The GameManager checks if the winning status is reached, monitors the turn switching during the whole game.

The AIManager is responsible to control the movements of shuttles. It gets decisions from the AIAgent and executes these actions on AI's turn. The AIBehavior includes the shuttle's properties.

The UIManager holds the timer, displays tips when switching turns and the end of the game.

Each tile of the board is attached to a TileController, which monitors human blocking on human's turn. Each counters generators are managed by a GeneratorManager which controls regenerations of pickups.

The Methods class contains several general methods and algorithms which are used by multiple classes. Some methods could be helpful for agent development. The detail describes in the API documentation.

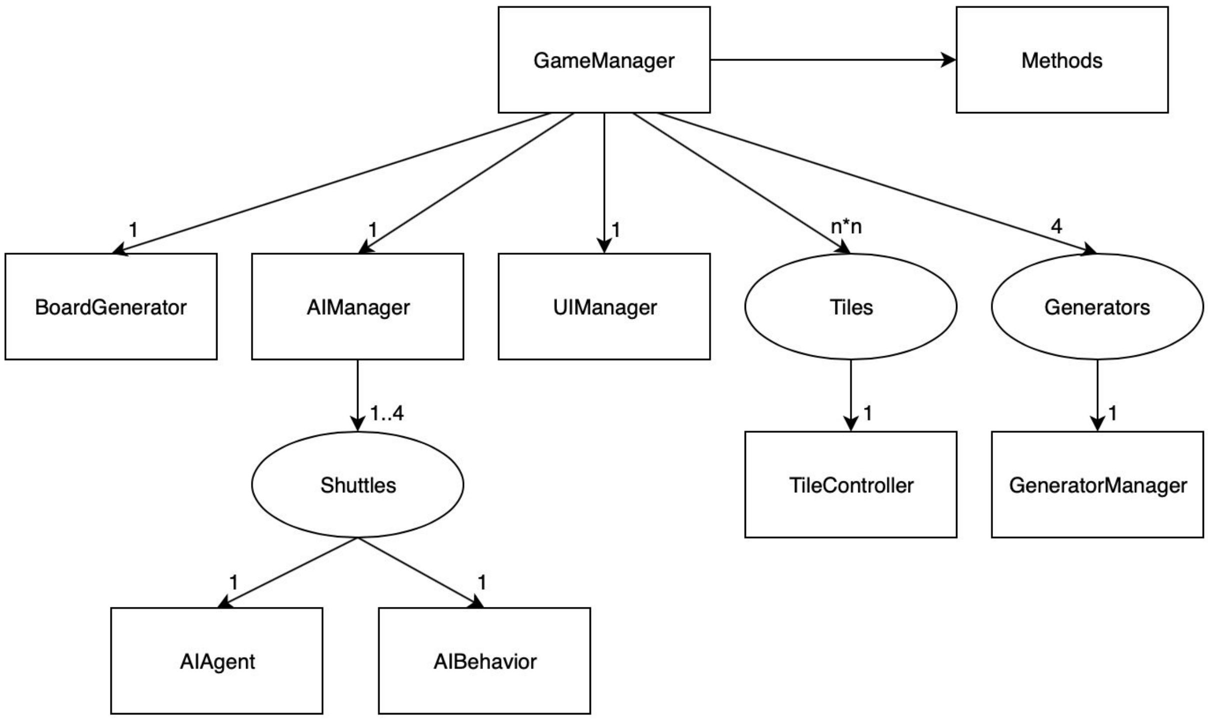


Figure 12. Main Components of the Game Scene

3.3 Design of the System

The class diagram of the menu scene is shown below. The MainMenu and the GameParameters are singleton classes. The HelpMenu is empty because the features are implemented in the Unity Inspector.

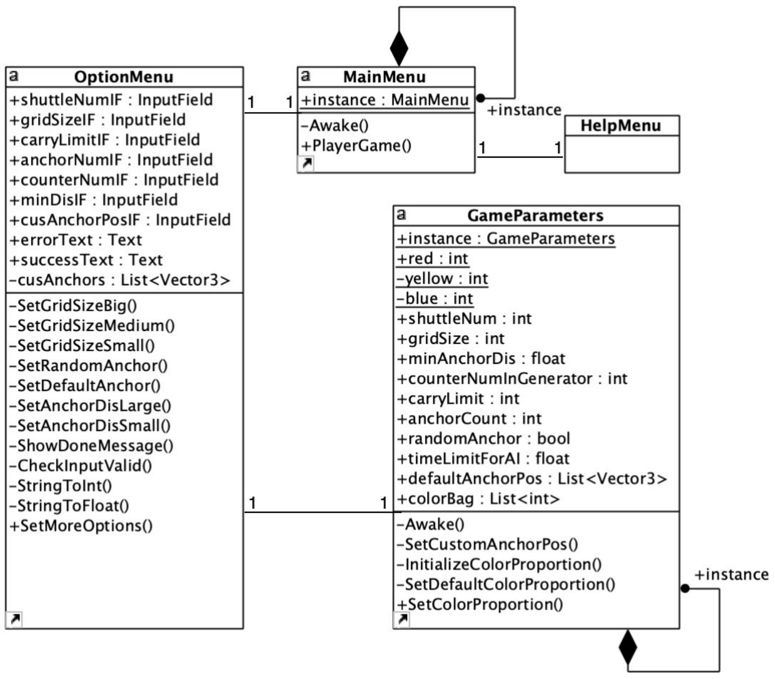


Figure 13. Class Diagram of the Menu Scene

The GameManager class and the Methods class are singleton in the game scene.

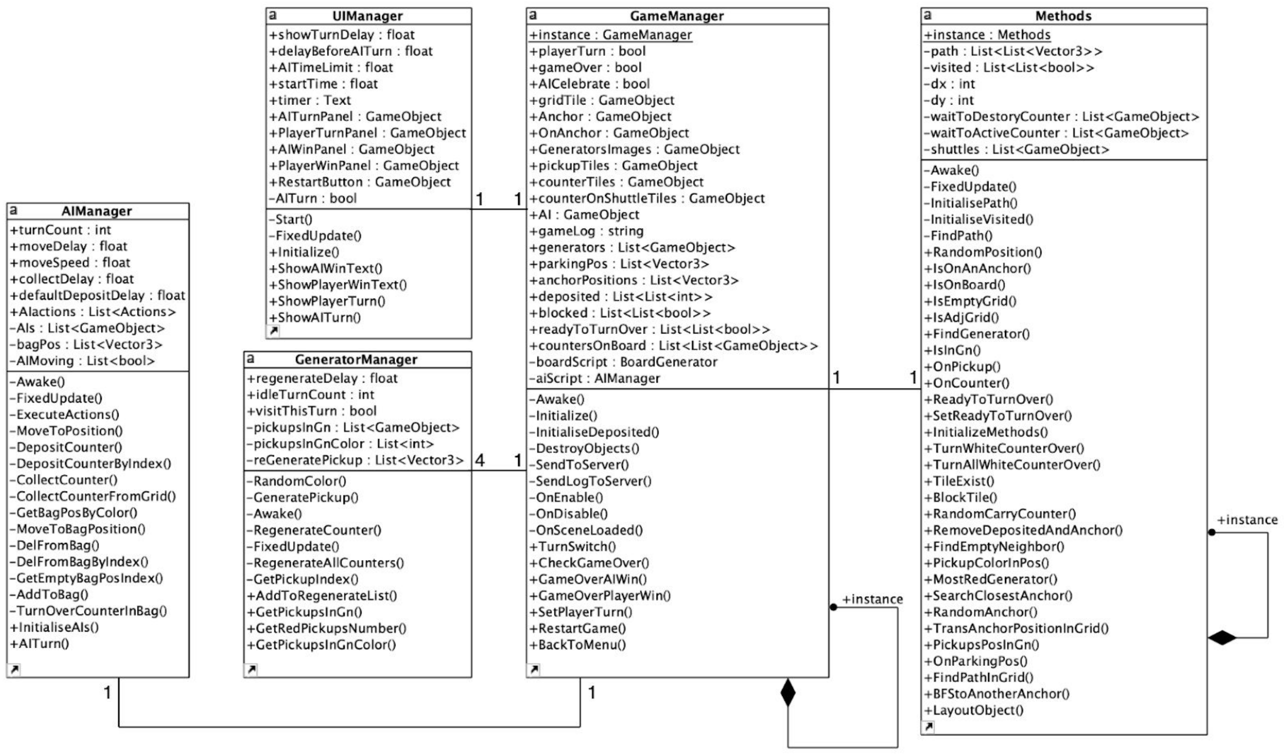


Figure 14. Main Class Diagram of the Game Scene

On AI’s turn, for each shuttle, the AIManager calls the “MakeDecision()” method in the AIAgent, then the AIAgent will return an “Actions” instance to the AIManager.

The Actions instance includes commands such as move, collect, deposit, turn over and corresponded parameters. The AIBehavior of each shuttle holds the information of the shuttle, such as how many counters remain in the shuttle and their colors. The AIBehavior also controls the logic of collision detection by “OnTriggerStay2D()” method and “OnTriggerExit2D()” method.

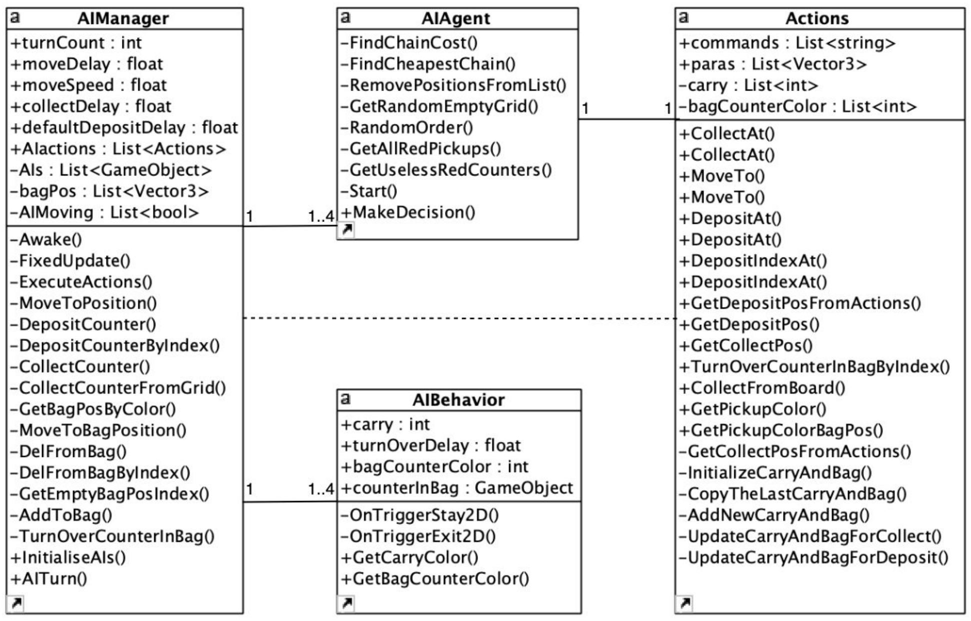


Figure 15. Class Diagram of the AI

3.4 Development process

3.4.1 Conceptual Model

This project is developed under Agile. In the first conceptual model, we developed a 3D version demo. In this version, we use a figure to represent the shuttle and a sheep to represent the anchor. The assets are downloaded from the Unity Asset Store ("Low Poly Game Kit," 2018). The human blockings are shown by cubes. However, after understanding the objective of the project, we plan to use 2D as it could ensure better performance on WebGL.

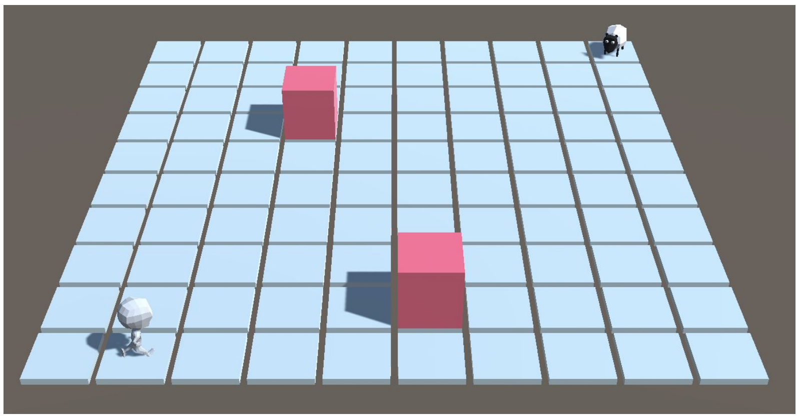


Figure 16. Conceptual Model

3.4.2 The First Sprint

In the first sprint, we implemented the basic functions of the game. The shuttle is a red cartoon figure and the anchors are treasure chests. These sprites are imported from "Free Platform Game Assets" (2019). The shuttle can only move in four directions and it is not able to across tiles blocked by the human player. The counters in the generators will be collected when they are collided by the shuttle.

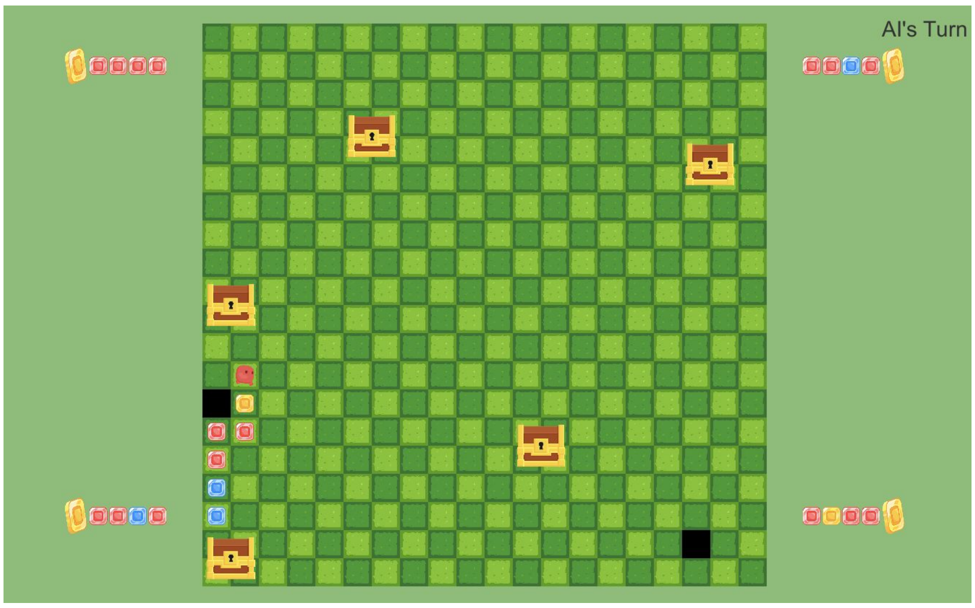


Figure 17. The First Sprint

3.4.3 The Second Sprint

In the second sprint, we added white counters and they could be turned over when the shuttle passes. Moreover, a vanilla agent is implemented, which will describe in section 5. The carrying information is displayed at the bottom of the screen.

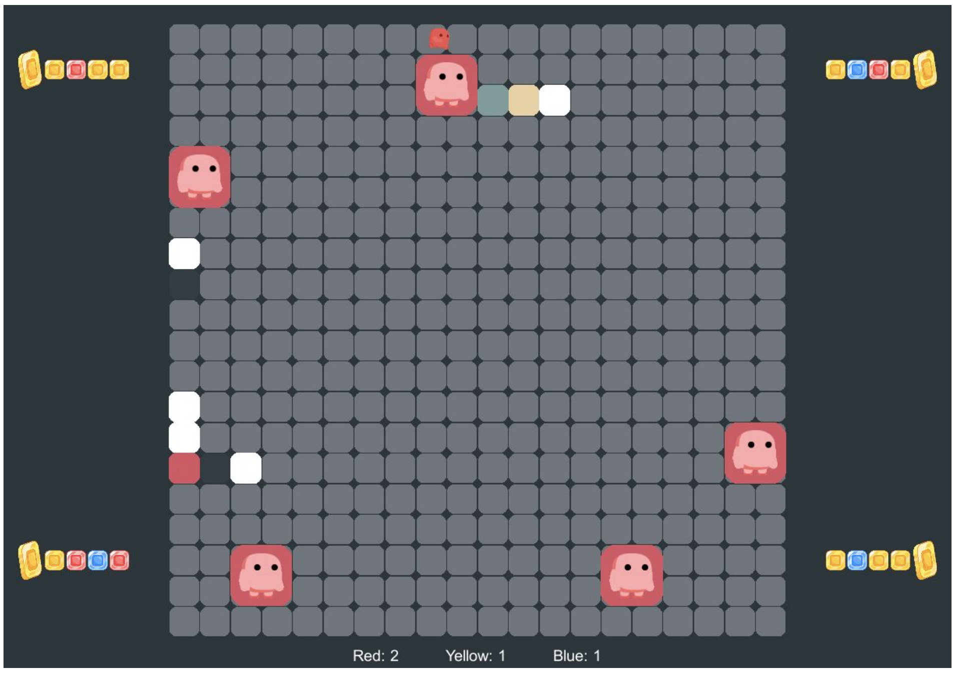


Figure 18. The Second Sprint

3.4.4 The Third Sprint

Next, we mainly implemented the visualization of the shuttle. The counters will appear in the shuttle after collected. As the shuttle occupied four squares, the collection became docking with the counters instead of collisions. After discussion, the shuttle became to move straightly. It ignored the grids and the blocked tiles won’t stop the shuttle anymore. Additionally, the overturned counters will turn back to white after a few seconds. All sprites after this version are original design using Photoshop.

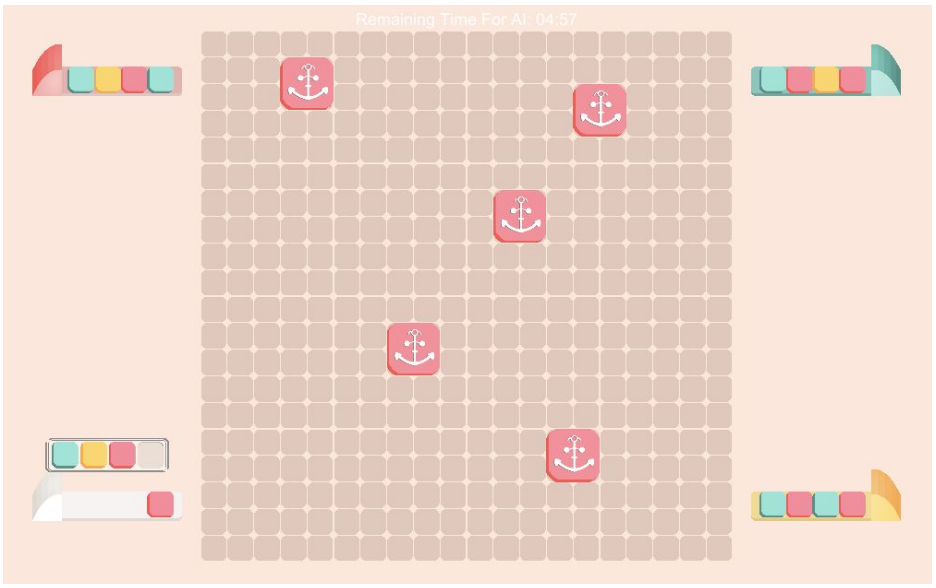


Figure 19. The Third Sprint

3.4.5 The Fourth Sprint

In the last sprint, we developed the menus, provided options and implemented an advanced AI agent. We also refactored the structure of the project and improved user experience such as providing the restart function. More APIs were supported such as turning over counters in the shuttle, collecting counters from the board.

3.5 Technology Choices

3.5.1 Collision detection

In this project, collision detection was used to detect counter collection and counter overturn on the board. In the beginning, we used the "OnTriggerEnter2D" to detect collision. It is a method provided by Unity and it will be called when the object enters another object which is attached a trigger collider ("Unity - Scripting API," 2019).

When the shuttle deposits a counter, it is actually destroying the counter object on the shuttle and creating the same counter on the board. In this case, the counters on the board are attached with trigger colliders and the method will be called when the shuttle enters. Therefore, we implemented "turn over" when the method was called. The problem was that we expected the counters to be face-down after deposit, but they could trigger the "OnTriggerEnter2D" and turn over immediately after deposit.

There are two solutions. The first one is recording the generation time of each counter and adding conditions before turning over. The difference between the current time and the generation time needs to be greater than a threshold. The defect is if the shuttle stays in the same place longer than the threshold, the counters will be turned over anyway.

The better solution is using the "OnTriggerExit2D" and the "OnTriggerStay2D" instead. The "OnTriggerExit2D" will be called when the object leaves a trigger collider and the "OnTriggerStay2D" will be called each frame where the two objects are overlapping ("Unity - Scripting API," 2019). Each counter is attached to a flag "readyToTurnOver", the flag is false by default and sets to true in the "OnTriggerExit2D". We turn over the counters in the "OnTriggerStay2D" if its flag is true.

However, the solution caused another problem. When a counter was turned over, it was actually hided the white counter and created a color counter there, then recovered the white counter and destroyed the color counter after a few seconds. We expected the counter to turn over for n seconds. If the shuttle stays on the counters, the counters should stay to face up.

In the test, although the counters stayed to face up, they would sparkle every n seconds. After debugging, we found that the issue was that the "OnTriggerExit2D" was not only decided by relative positions, but also awaken before the object was destroyed. Therefore, the color counter would be destroyed every n seconds, then triggered "OnTriggerStay2D" and generated another color counter, which causes the sparkle issue.

The final solution is adding conditions when destroying the color counter. If it still collides with one shuttle, add it to a "waitToDestoryCounterList" and add the corresponding white counter to a "waitToActiveCounterList", then process these counters after the shuttle leaves in the "FixedUpdate" function ("Unity - Scripting API," 2019) which is called every fixed frame-rate frame.

As the collection method became docking with the counters, it doesn't use collision detection in the final version. Therefore, although the counters in the generators and the counters on the boards share the same appearances, they are different prefabs in the assets called "pickups" and "counters" respectively. The collection logic is controlled by commands and the collisions won't be caught between pickups and shuttles.

3.5.2 Turn switch

The "Update()" is a commonly used method provided by Unity. It will be called every frame ("Unity - Scripting API," 2019). In a large number of tutorials, "Update()" is used to update the game states ("Unity Learn," 2019). In the first few versions, at every frame, we used "Update()" to detect whose round is currently. Setting the turn switch condition is difficult. We decided to switch the turn when the shuttle returns to the board after collection, which is the shuttle is out of the board in the last frame and within the board in the current frame. However, the shuttle might collect a few counters in the left side generators firstly, then collect other counters from the right side generators on the same turn. In this case, the shuttle will be judged as "return to the board" twice and switch to human's turn twice.

After analysis, we found the "Update()" method is not suitable for the turn-based game. Therefore, in the final version, we generally switch the turn when the AIManager has executed all Actions of all shuttles. The Action limitation in each turn needs to be controlled by the AIAgent.

3.5.3 Log File

To collect data, a log file will be sent to the server. The project is published by WebGL which cannot access local files for security reasons. To overcome this issue, we record the game log in a string in the GameManager. At the end of each game, we call a simple PHP file on the server and sent the log string to it, then the PHP file will write the string to a text file on the server (Holistic3d, 2018).

3.5.4 AI Agent Decision

In the first a few versions, the "MakeDecison()" method only decide the next move at a time and it would be called multiple times on each AI's turn. For example, to deposit a counter at (0,0), the agent firstly commands the shuttle to collect a counter when calling the "MakeDecision()" for the first time, then commands to move to (0,0) at the second time, finally, deposits at (0,0) at the third time.

Making a decision at each move could be useful in some interactive games such as Pac-Man, as the states of pacman and ghosts are changing at every frame. However, in this game, the human player only can make a move after the AI has done all actions on its turn. In this case, making decisions for the whole turn each time could be easier to control the overall strategies. Therefore, it became the choice we applied in the final version.

4. API Documentation

4.1 Overview

This section contains the core content of the scripting API that the game provides to support AI agent development. We listed a few examples below. The full documentation is available on Github.

4.2 Actions

Before each turn, the AI agent needs to create an Actions class which includes all actions the shuttle(s) will take sequentially in this turn. The following methods help to give the shuttles orders.

Public Functions	Description
CollectAt	Collects pickup(s) from position(s)
MoveTo	Moves to position(s)
DepositAt	Deposits a specified colored counter at the position, sets delay optionally
DepositIndexAt	Deposits a specified indexed counter at the position, sets delay optionally
GetDepositPos	Returns a list of positions that have been added deposit commands in this actions as well as all other shuttles' actions.
GetCollectPos	Returns a list of positions that have been added collect commands in this actions as well as all other shuttles' actions.
TurnOverCounterInBagByIndex	Turns over the counter with the given index
CollectFromBoard	Collects a counter from the board
GetPickupColor	Returns how many red, yellow and blue counters are carried according to the Actions
GetPickupColorBagPos	Returns the color of counter on each bag position

4.2.2 Actions.CollectAt

public void CollectAt(List positions);

Parameters	Description
positions	The list of pickups' positions that needs to be collected

Description

Collects a list of pickups from positions. A shuttle should move to the corresponding parking position of the generator before collecting pickups from it. All positions from the list need to be valid exist pickups' positions from the same generator.

---

public void CollectAt(Vector3 pos);

Parameters	Description
pos	The position that needs to be collected

Description

Collects a pickup from the position.

4.2.3 Actions.DepositAt

public void DepositAt(Vector3 pos, int color);

Parameters	Description
pos	The pos that needs to deposit
color	The counter's color

Description

Deposits a specified colored counter at pos. A shuttle should move to pos before depositing.

---

public void DepositAt(Vector3 pos, int color, float delay);

Parameters	Description
pos	The pos that needs to deposit
color	The counter's color
delay	The delay before deposit

Description

Waits delay seconds then deposits a specified colored counter at pos. A shuttle should move to pos before depositing.

4.3 GameManager

GameManager holds the states of the game, includes generators, anchors, deposit records and block records.

Variables	Description
generators	The GameObjects of all generators
parkingPos	The parking position of all generators, the shuttle needs to move to the parking position before collect
anchorPositions	The positions of the centre of anchors
deposited	The deposit status of each grid
blocked	The block status of each grid

4.4 GeneratorManager

Every generator corresponds to a GeneratorManager which controls pickups' generations.

Variables	Description
regenerateDelay	The delay before regenerating a pickup after collecting

Public Functions	Description
GetPickupsInGn	Returns a list of GameObjects of all pickups in the generator
GetRedPickupsNumber	Returns the number of red pickups in the generator
GetPickupsInGnColor	Returns the color of the pickup at the given position

4.5 AIAgent

Every shuttle instance attaches an AIAgent and an AIBehaviour. AIAgent is where users define their own agents.

Public Functions	Description
Start	Called on the frame when a script is enabled
MakeDecision	Called before each AI's turn

4.6 AIBehavior

AIBehavior includes AI's properties such as information about carrying counters.

Variables	Description
turnOverDelay	Turns over the counters how many seconds when the shuttle collides with them

Public Functions	Description
GetCarryColor	Returns the number of each counter
GetBagCounterColor	Returns the color of each counter in the shuttle's bag

4.7 GameParameters

GameParameters control the parameters of the game, includes the size of the board, the number of the shuttles, the minimal distance between two anchors, the number of counters in each generator and so on. If users make selections in the "Options" menu, it will cover the values in GameParameters.

Variables	Description
shuttleNum	The number of shuttles
gridSize	The default size of the board
minAnchorDis	The minimal distance between two anchors

counterNumInGenerator	The number of counters in each generator
carryLimit	The number of counters each shuttle can carry
anchorCount	The number of anchors
randomAnchor	The positions of the anchors are random or default
timeLimitForAI	The time limit for AI

Functions	**Description**
SetCustomAnchorPos	Edit this function to set customized anchor positions
InitializeColorProportion	Edit this function to initialize the proportions of each colored counter in the generators
SetColorProportion	Sets the proportions of each colored counter in the generators during the game

5. Agent Development

5.1 A Vanilla Agent

The vanilla agent is straightforward. At the beginning of each game, the agent chooses the nearest two anchors and uses Breadth-First-Search to calculate the shortest path between them as the true path. In addition, it will randomly choose two anchors as a fake path. On each turn, the shuttle deposits some of the yellow or blue counters on the fakepath orderly, while deposits red counters randomly on the true path. There is a possibility that the shuttle may deposit rest of yellow or blue counters around the true path.

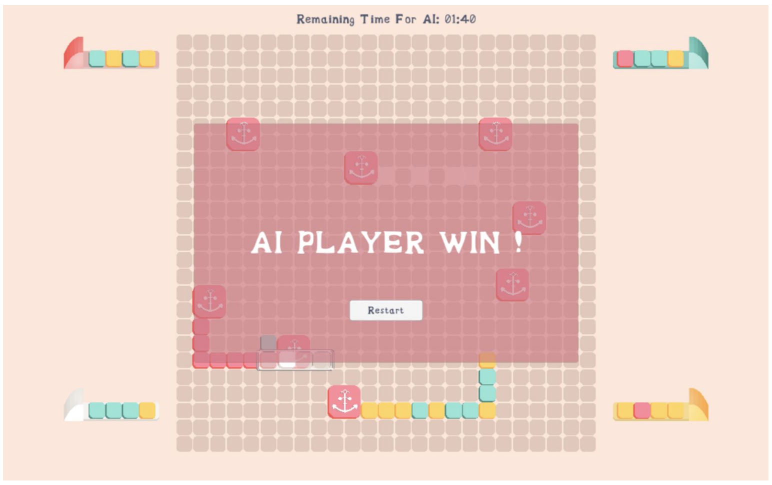


Figure 20. A Game Result Using the Vanilla Agent

To pretend consideration, the shuttle will wait a few seconds before deposit a fake counter. The deposit API allows setting a delay optionally.

5.2 An Advanced Agent

The cost between two anchors is defined as the minimum manhattan distance between them minus the number of grids which have been deposited red counters on the path. At the beginning of each turn, the advanced agent will randomly choose a true path from the least cost and the second least anchor pairs. Next, it chooses the least cost anchor pair except for the true pair as the fakepath.

The agent will deposit most of the red counters on the true path and deposit others on the fakepath. It will wait a random seconds before any deposit. There are other strategies, for example, when the shuttle deposits a counter, it will randomly turn over another counter as interference. If there are multiple shuttles, the shuttles could consider the actions of other shuttles and avoid to collect or deposit at the same positions with others.

Before each turn, the agent will firstly check whether it can win in this turn, which is building a connected chain of red counters between any two anchors within four depositions for each shuttle. If it is a winning state, the agent will collect all required red counters from all generators as well as the board, then deposit them on the least-cost path.

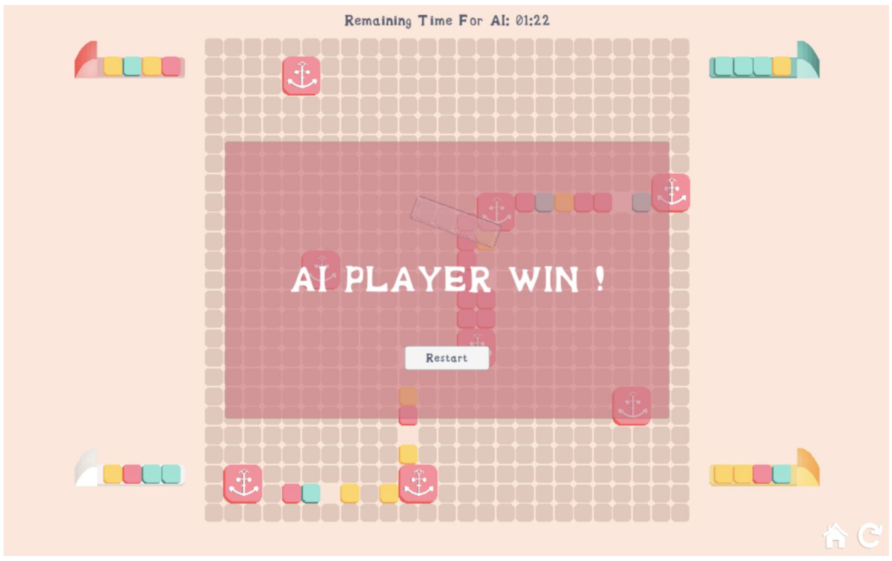


Figure 21. A Game Result Using the Advanced Agent

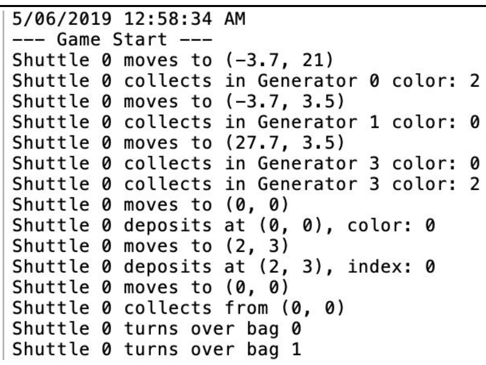
6. Test

6.1 APIs Test

We use a simple script to test the main APIs for agent development, then use the log file to prove the results. The shuttle will firstly collect one counter from generator 0 and generator 1 respectively, then collect two counters from generator 3. Next, the shuttle will deposit a red counter at (0, 0), then deposit the first counter on the shuttle at (2, 3). Moreover, it will collect the counter at (0, 0). Finally, it turns over the first and second counters on the shuttle.

The full test script could be found in “DeceptionGame/OtherScripts/AIAGent_test.cs”. The main component is shown below:

```
        // Gets 1 position of pickup from generator 0
        List<Vector3> collectList = Methods.instance.PickupsPosInGn(0, 1);
        // Moves to the parking position of generator 0
        actions.MoveTo(GameManager.instance.parkingPos[0]);
        // Collect 1 pickup from generator 0
        actions.CollectAt(collectList);
        // Gets 1 position of pickup from generator 1
        collectList = Methods.instance.PickupsPosInGn(1, 1);
        // Moves to the parking position of generator 1
        actions.MoveTo(GameManager.instance.parkingPos[1]);
        // Collect 1 pickup from generator 1
        actions.CollectAt(collectList);
        // Gets 2 positions of pickups from generator 3
        collectList = Methods.instance.PickupsPosInGn(3, 2);
        // Moves to the parking position of generator 3
        actions.MoveTo(GameManager.instance.parkingPos[3]);
        // Collect 2 pickups from generator 3
        actions.CollectAt(collectList);
        // Moves to (0, 0)
        actions.MoveTo(new Vector3(0, 0, 0));
        // Gets the number of red counters on the shuttle now
        int redNum = actions.GetPickupColor()[0];
        // Deposits a red counter at (0, 0) if has one
        if (redNum > 0)
        {
            actions.DepositAt(new Vector3(0, 0, 0), 0);
        }
        // Moves to (2, 3)
        actions.MoveTo(new Vector3(2, 3, 0));
        // Gets the color of each bag on the shuttle now
        int[] bag = actions.GetPickupColorBagPos();
        // Deposit the first counter at (2, 3) if the counter is available
        if (bag[0] != -1)
        {
            actions.DepositIndexAt(new Vector3(2, 3, 0), 0);
        }
        // Moves to (0, 0)
        actions.MoveTo(new Vector3(0, 0, 0));
        // Collect the counter at (0, 0)
        actions.CollectFromBoard(new Vector3(0, 0, 0));
        // Gets the color of each bag on the shuttle now
        bag = actions.GetPickupColorBagPos();
        // Turns over the first counter on the shuttle if the counter is available
        actions.TurnOverCounterInBagByIndex(0);
        // Turns over the second counter on the shuttle if the counter is available
        actions.TurnOverCounterInBagByIndex(1);
```



```
5/06/2019 12:58:34 AM
--- Game Start ---
Shuttle 0 moves to (-3.7, 21)
Shuttle 0 collects in Generator 0 color: 2
Shuttle 0 moves to (-3.7, 3.5)
Shuttle 0 collects in Generator 1 color: 0
Shuttle 0 moves to (27.7, 3.5)
Shuttle 0 collects in Generator 3 color: 0
Shuttle 0 collects in Generator 3 color: 2
Shuttle 0 moves to (0, 0)
Shuttle 0 deposits at (0, 0), color: 0
Shuttle 0 moves to (2, 3)
Shuttle 0 deposits at (2, 3), index: 0
Shuttle 0 moves to (0, 0)
Shuttle 0 collects from (0, 0)
Shuttle 0 turns over bag 0
Shuttle 0 turns over bag 1
```

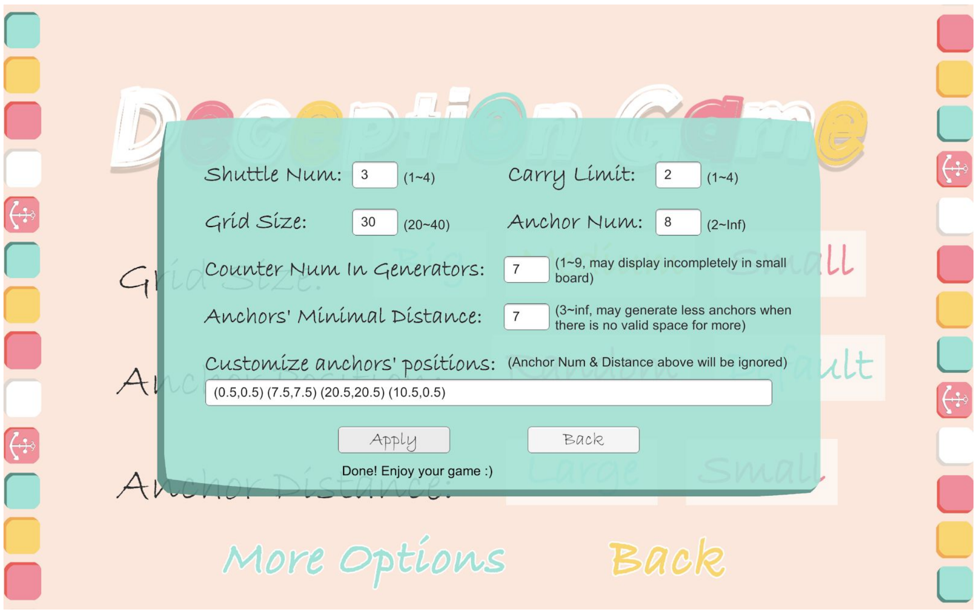
Figure 22. The Log File of Test

The log file is in Figure 22. The test shows that the main APIs worked well.

However, the test might fail in some cases, for example, when there are an anchor at (0, 0) or (2, 3) or the shuttle didn't hold any red counter after collections.

6.2 Options Test

In this section, we plan to test the custom functions by specifying the parameters of the game. In the "More Options", we set the parameters as following:



Shuttle Num: 3 (1~4)
Carry Limit: 2 (1~4)
Grid Size: 30 (20~40)
Anchor Num: 8 (2~Inf)
Counter Num In Generators: 7 (1~9, may display incompletely in small board)
Anchors' Minimal Distance: 7 (3~inf, may generate less anchors when there is no valid space for more)
Customize anchors' positions: (Anchor Num & Distance above will be ignored)
(0.5,0.5) (7.5,7.5) (20.5,20.5) (10.5,0.5)
Done! Enjoy your game :)

Figure 23. Set the Parameters

The screenshot shows there are three shuttles with 2 carry limit, the board size is 30, each generator holds 7 counters and the anchors are distributed at the right positions.

The anchor number and the minimum distance are ignored because the anchors' positions are specified.

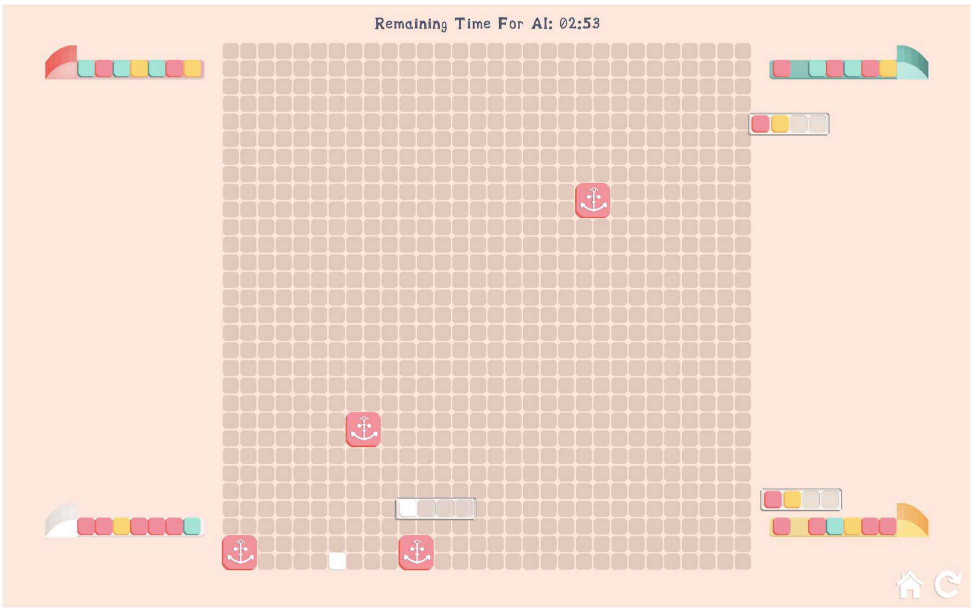


Figure 24. The Result of Options Test

7. Future Work

There are other strategies may use in future development, such as considering the human reactions: where they block, how much time did they cost to think. There are some APIs we didn't use in the two agents described in section 5. For example, there is an API "SetColorProportion" which can control the color proportion of counter generators without notifying human player.

Adding audios and animations might be helpful for the "magic tricks", which could be used to misdirect attentions.

Additionally, this deception game needs to be available on Mechanical Turk in the future, which could be convenient for running experiments and collecting data.

8. Conclusion

In this project, we built a deception game and provided APIs to support future agent development. We also tried to find that how AI might deceive people and two simple agents are developed as a baseline. We also believe that this small project has helped the deceptive AI project find a clear direction.

This project is developed under Agile. We negotiated the requirements with the supervisors as well as the clients. At the beginning, some requirements are ambiguous. Moreover, we also misunderstood several features, which caused many reworks. As only some of the features are implemented in each sprint, there is inadequate consideration of the structure of the whole system. This problem has been solved after multiple refactorings. The total development time is about 155 hours.

During this project, we learned to use Unity, C# and improved communication skills. Unity is a powerful game engine which could support all of the development requirements. However, building a WebGL project by Unity still has many compatibility limitations. For software development, we learned it is important to consider the code reusability at the beginning.

Reference

Deceptive AI. (n.d.). Retrieved from https://cis.unimelb.edu.au/research/groups/interaction-design/projects/deceptive-ai/

Free Platform Game Assets. (2019, Apr). Retrieved from https://assetstore.unity.com/packages/2d/environments/free-platform-game-assets-85838

Holistic3d. (2018, March 28). How to Read and Write to a Textfile from Unity WebGL Part 1 [Video file]. Retrieved from https://www.youtube.com/watch?v=4OZqY1Ukj8I

Low Poly Game Kit. (2018, Aug). Retrieved from https://assetstore.unity.com/packages/templates/packs/low-poly-game-kit-110455

Unity Learn. (2019, Jun). Retrieved from https://learn.unity.com/tutorials

Unity - Manual: 2D or 3D projects. (2019, Jan). Retrieved from https://docs.unity3d.com/Manual/2Dor3D.html

Unity - Manual: Unity User Manual. (2019, Jan). Retrieved from https://docs.unity3d.com/Manual/index.html

Unity - Manual: WebGL Browser Compatibility. (2019, Jan). Retrieved from https://docs.unity3d.com/Manual/webgl-browsercompatibility.html

Unity - Multiplatform: Publish your game to over 25 platforms. (2019). Retrieved from https://unity3d.com/unity/features/multiplatform

Unity - Scripting API. (2019, Jan). Retrieved from https://docs.unity3d.com/ScriptReference/