**Directory**

[1 Introduction 2](#_Toc16885)

[1.1 Background 2](#_Toc11146)

[1.2 Aim and Objectives 3](#_Toc9873)

[2 Related Work 4](#_Toc32478)

[2.1 Mobile game engines 4](#_Toc1982)

[2.2 The “Match Three games” 5](#_Toc11767)

[3 Technical development 7](#_Toc22065)

[3.1 The game flow design 7](#_Toc13440)

[3.2 The basic initialization of the game 10](#_Toc23948)

[3.3 The basic algorithm of the game 11](#_Toc31666)

[4 Implement and Result 15](#_Toc26067)

[4.1 Implementation 15](#_Toc26725)

[4.2 Figures of Game Functions 23](#_Toc12739)

[5 Conclusion 28](#_Toc26270)

[6 References 28](#_Toc18585)

# 

# Introduction

## 1.1 Background

With the rapid development of the internet and improving development technologies of smartphones in the recent years, many smart phone games such as ‘Angry Birds’, ‘Temple Run’, have achieved great successes. A large amount of individual developers or independent develop teams have earned their first pot of gold.

Advanced mobile techniques can lead to mobile games are accounting for a majority of global games market (Yi, Lee & Kim, 2017). A trend has been found out that the number of casual mobile games based on the mobile platform is increasing rapidly, especially for the casual “Match Three” games. For example, the famous casual “Match Three” games such as the original Bejeweled, Candy Crush Saga and Evolve: Hunters Quest all attract millions of downloads and players.

Mobile game market is unprecedentedly prosperous. And there is a huge space for development and well profit prospects. Netease, Tencent, Giant, Kai Ying network and other well-known companies in the industry in China changed the focus of the game business to the mobile platform.

As a result, the title of my project is a mobile casual “Match Three” game based on the android system. The main purpose of my project is to build a game that incorporates both 2d casual “Match Three” game and 3d Parkour game. It will be deployed on the android system. Unity is a mature software which has developed many distinguished types of games such as “Match Three” game, puzzle game, and Parkour game.

This report will show the progress of the project in five main sections. Firstly, it will give an introduction of the project and the aims and objectives I want to achieve in this project. Secondly, I will demonstrate the related research, which are “Match Three” games and the background of mobile game engine. Next, I present the technical development on the game flow and discuss the complete game design including initialization and the algorithms in details. In addition, I show the implementation and some function results. Finally, I will make a conclusion and summarize the project.

## Aim and Objectives

The overall aim of my project is to build a casual “Match Three” game and implement on Android system.

To achieve this aim I need to divide this project to several objectives as below.

Objective 1 – Using Unity 2D to build a 2D game scene

1.1 Download and install Unity 2D

1.2 Using Unity 2D to build a 2D game environment

1.3 Add some elements into the 2D game environment

1.4 Modify the parameters of elements

1.5 Test the 2D game scene and debug

Objective 2 – Designing 2D model and UI

2.1 Import 2D models into Unity 2D

2.4 Set the parameters of 2D models

2.5 Design the 2D UI of the game

2.6 Debug and modify 2D UI of the game

Objective 3–Writing C# script, algorithm and test and debug the game

3.1 Design algorithm of the game.

3.2 Using C# to write script.

3.3 Implement script and debug.

3.4 Overall test and debug.

Objective 4– Deploy the game on Android.

4.1 Prepare Windows and android devices.

4.2 Use Unity 2D to transplant the game to the android device.

4.3 Run the game on android device.

* 1. Test and debug the errors in Unity 2D.

# Related Work

## 2.1 Mobile game engines

In the recent years, three-dimensional games on the mobile platform have been the main trend of game business. “With the development of Computer technology, it promotes the growth of a range of related industries. Mobile phone is very popular in our society, and the trend of that more people will play mobile game in mobile phone is more and more obvious.” (Jiang, Kuang, Shen, 2011,p.317)

There are many game engines for mobile platform. Game engine is the key to develop games. Unity is a cross-platform game engine that can be used to develop standalone games on Windows, MacOS, Linux, or device games on iOS, Android mobile. Unity can also develop web games that support WebGL technology, or games on PlayStation, XBox, Wii hosts. Unity is a really powerful, highly integrated and easy to use. A large amount of mature technologies have been used on Unity to help developers building games (Blackman, 2013; Lu et al., 2011).

In addition, a language relevant to game development is C#. C # is a secure, stable, simple, elegant, object-oriented programming language which derived from C and C ++. It inherits C and C ++ powerful features while removing some of their complex features (such as no macro and does not allow multiple inheritance). C # combines VB's simple visualization and high operating efficiency of C ++, and is the preferred language for .NET development with its powerful operating power, elegant grammar style, innovative language features and convenient component-oriented programming support (Norton, 2013; Smith, 2013).

## 2.2 The “Match Three games”

In recent years, the type of *Match Three* games (or called as bubble birzzle games) is pretty famous on the market, especially among children, teenagers and young ladies for its cute pictures and easy handling. The famous games such as the original [Bejeweled](http://www.windowsphone.com/el-gr/store/app/bejeweled-live/ea9a24ad-d2d1-df11-9eae-00237de2db9e?signin=true), [Candy Crush Saga](http://www.windowsphone.com/el-gr/store/app/candy-crush-saga/aebbdef8-1792-488b-a7be-7596a1720166) and [Evolve: Hunters Quest](http://www.windowsphone.com/en-us/store/app/evolve-hunters-quest/82a8028d-82e3-4897-97e0-3389f6a0e802) all attract millions of downloads and players. The success of these games lies in the smart combination of the basic mechanism of Match Three and the specialized bonuses given to their players.

The rule or goal of a typical Match Three game is quite simple, i.e. to swap pieces of items around until there is a three or more (up to five) identical items in a row or column (generally, there are four to five sorts of items in one game). When such a match is formed, those bubbles are eliminated and then the empty locations left will be filled. For a special move or swap that matches more than three items in a row or line, the match will offer the swapped item a special bonus. Generally, the bonus is different with different match: the swap that matches four items in a row or column will earn a bonus that has the power to eliminate a row of column, respectively; the swap that matches a “T” type double three items will earn a boom that can eliminate the surrounding 25 items; and the swap that matches five identical items in a row or column will earn the very special bonus, such as a chance to eliminate all the items of one kind. Each match or elimination will earn player many points, the special elimination and potential combos will double or triple these points. The total points in a game can be uploaded to the server and yields a ranking list among all players or familiar players, which also results in the competing willing and enthusiasm of “winning” among players to keep playing this game.

Today, the casual Match Three has become one of the most popular games in the mobile game market, which will maintain the foreseeable attractions as well. This kind of games is extremely simple to learn and play, takes short time yet bring users much joy during when taking the bus/subway or waiting for a cup of coffee. Hence, for smart phone owners, the Match Three games are quite suitable for them, since mobile platform and the fragmented time do not match with the “traditional”, long-time and complex games (Julkunen, 2015). In addition, the Match Three game has quite a high level of randomness (or at least the randomness displayed on the screen), which brings players the freshness even in the repeated game session. Moreover, the short game time of Match Three can output the instantaneous feedback of quick success or failure, which hence creates an addictive combination of strong feelings with quick shots of joy or anger into players’ nervous system, and then attracts or even “traps” its players.

The mechanism of the Match Three games brings a huge success in the mobile game market. Take the famous game Candy Crush Saga as an example. As one of the most popular mobile games in the U.S. market, crossed a total of 2.73 billion sugary downloads at its fifth anniversary in the fourth quarter in 2017 (Takahashi, 2017). At the end of the third quarter in 2017, this Match Three game earned more than $250 million in revenue, which is more than 97% higher than the same period in 2016, based on the measurement of firm Sensor Tower, as shown in Figure 4-1 (Takahashi, 2017). Candy Crush Saga started from the platform on Facebook, but took off in the mobile market. Since its launch, the game has more than 2,800 levels, more than 1 trillion game rounds has played in the game, 350 trillion candies have been swiped to date, cultivated a large social media fan circle of 80 million, and has been ranking the top ten grossing apps for four years now (Takahashi, 2017).

# Technical development

## 3.1 The game flow design

Taking the Candy Crush Saga as an example, the basic game flow of a typical Three Match game is illustrated in Figure 4-2, which consists of the following steps.

1. The player drags an item and attempts to swap it with another one surrounding.
2. An auto-check is operated to judge whether the swap can yields any three or more identical items in a row or column.
3. If the check is failed, the swap will be undone.
4. If the check is passed, the marched items will be removed and leads to the following operation automatically.
5. The items over the matched and removed items will drop down towards to the bottom of the screen to fill the blank locations used to be occupied by those removed items, due to the gravity effect.
6. During the drops of these items (or called as the collapse process), the auto-check will be activated again to judge whether a new three or more matched items will form in the new row or column. If the check is failed, the drop will continue to the upper row. But if the check is passed, the newly formed matched item will be removed automatically and leaves additional blanks. This kind of dropping, checking and removing will continue row by row until the first row (the top of the screen).
7. Then new items will drop to the top row randomly to make up the whole game region, and again, the auto-check is activated judge whether new matched items are formed. If the result is “No”, then the collapse process is ended; whereas if the result is “Yes”, then the same process from step 6 to step 7 will be activated repeated automatically until the auto-check in step 7 returns a “No” response.
8. Then it is back to the player’s turn to make another new swap attempt.

To make the game more interesting, bonus system is generally embedded in as follows. The bonus will be yield immediately when a more than four matched items are formed, either resulted from the drag or swap of the player, or from the collapse process automatically. There are fore different types of bonus.

1. If a swap of a drop results in a four matched items in a row or a column, then the very one that yields the match will be gift a bonus property that can remove the whole row or column in the next three match (i.e. if the item containing the bonus forms a new three or more match, the whole row or the column where the bonus one locates will be removed entirely), and the bonus one will be removed after the swap as well.
2. If a swap of a drop results in a “T” type double three items, then the very one that yields the match will be gift a bonus property that can remove its surrounding five times five items in the next three match as a “boom”, and the bonus one will be removed after the swap as well.
3. If a swap of a drop results in a four matched items in a row or a column, then the very one that yields the match will be gift the biggest bonus. The bonus one will be regarded as the special one that does not belongs to any item sort in the game. Any swap of the bonus one to the common one (belongs to any item sort of the game) from the player will remove all the existed “same-sort” items in the entire game, and the bonus one will be removed after the swap as well.

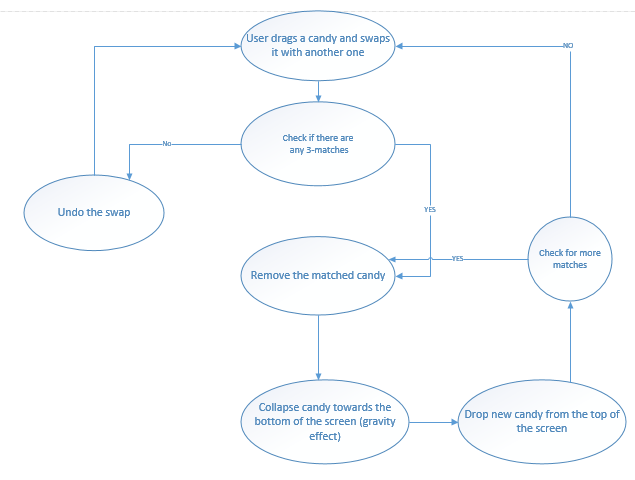


Figure 4-2 Illustration of the basic game flow of a typical Three Match game

## 3.2 The basic initialization of the game

The basic initialization of the game contains two aspects. One is the initialization related to the items, and the other is the initialization related to the game board.

3.2.1 The initialization of the game board

The initialization relates to the game board includes the size of the game board, scores, animations, sounds, durations and other elements. These elements can be generated and described through the Constants class and sound manager.

For Contains class, it contains some basic self-explainable variables some and the essential useful constants for the initialization of the Match Three game.

3.2.2 The initialization of the items

For each item appearing in the game board, a Shape class is used to describe the essential properties of these items. For each item GameObject shown on the screen, a Shape component is defined to contain information on the Type of the Shape (which not only contains the outlooking of the item, but also works as the class id for the match check), the position variable describing the item location (i.e. a two dimensional array contains the Column Number and Row Number), potential bonus of the item, a method used to make a comparison between the current item Shape with surrounding one.

## 3.3 The basic algorithm of the game

The Match Three game has two basic states, the initial state and the gaming state, which has different basic operation mechanism and algorism. For the initial state, there is a basic algorithm for the initialization of the game board. For the gaming state, the basic algorisms deals with the match check, the collapse process, and the hint function.

3.3.1 The algorithm for game board initialization

For the initialization of the game board, there are two important principles: (1) there should be no three or more matches in the initialized game; and (2) there should be at least one potential swap to form three or more matches in the initialized game, i.e. “none swappable state” is not allowed.

For the first principle, the following algorithm is applied to avoid the three or more matches initialized in the game board.

The item is generated and filled in the board from the bottom to the top and from the left to the right, as shown in the table 4-1. The symbol map containing six sorts of symbols are used in the table to represent the game board and the items. The black “#” symbol represents the blank positions waiting to be filled in during the initialization of the game board, and as introduced above, the item is generated from the bottom to the top and from the left to the right. The blue “!” “/” “]“ “;” and “}” symbols represent the generated items, with one sort of symbol representing one type of item. The red “?” symbol represents the current position waiting to be filled in with optional items. For the first two columns, when filling in the current position, it is needed to consider the types of the two items below the current position. If the two items below the current position belong to two different types, then any kind of item can be filled in the current position; whereas if the two items below the current position belong to one identical type, then the same type of item must be excluded from the choices. When filling the third and the following columns, both the two items below and two items locating left to the current position need to be analyzed. If the two items below the current position belong to one identical type, or the two items locating left to the current position belong to another (or the same) identical type, then these types of items can not be filled in the current position in case to generate a game board with a “match three items” initialized at the beginning.

Table 4-1 Description on the initialization of the game board with different types of symbols representing different item types, except that the # and ? represent the blank and current position waiting to be filled in, respectively

|  |  |  |
| --- | --- | --- |
| First column | First two columns | Three and more columns |
| # # # # # #  # # # # # #  ? # # # # #  ! # # # # #  ! # # # # #  / # # # # # | ; # # # # #  ] # # # # #  ] # # # # #  ! ? # # # #  ! } # # # #  / } # # # # | ; / # # # #  ] / # # # #  ] ] ? # # #  ! ; ; # # #  ! } ; # # #  / } ! # # # |

For the second principle, the following algorithm is applied to avoid the initialized game board containing no potential swap to form the “Three Matched items”, i.e. to avoid the “none swappable state”.

As shown in the Figure 4-2, if the game board containing the following three kinds of configuration, then the game has the potential swap to form the “Three Matched items”. The first is that the game board contains at least two identical items next to each other in the same column (as the red boxes marked as “1” and “2” in the left region of Figure 4-2, respectively), and contains at least one identical item locating at the corner of the two neighboring items, as represented by the four yellow boxes in the left region of Figure 4-2. Similar state shown in the middle region of Figure 4-2 also has the potential swap to form the “Three Matched items”, as long as the yellow corners contain at least one same item as the neighboring two items in the red boxes marked “1” and “3” in the same row. The last state that has the potential swap to form the “Match Three items” is shown in the right region of Figure 4-2. If at least one of the four groups of yellow boxes, “a and b”, “b and c”, “c and d” and “a and d” contains the same items as the red boxes marked “1”, then the game board has the potential swap to form the “Three Matched items”

If the game board containing no potential swap to form the “Three Matched items” after the analysis above, there are many ways to make a small modification on the generated configuration, and one of the simple ways is as follows. During the analysis process, it can be figured out whether the current game board configuration has at least two identical items next to each other in the same row or in the same column. If there are two identical items next to each other, then switching any item locating at the four corners of the two neighboring identical items to the same one, then the configuration will become “swappable”. While if the current configuration has no identical items next to each other, then a configuration same to one of the three configurations introduced above can be constructed in any region of the game board.

图片包含 障子, 文字, 建筑物

描述已自动生成

Figure 4-2 Illustration of the three states of the game board that has the potential swap to form the “Three Matched Items”

* + 1. UML

System Sequence Diagram

图片包含 文字

描述已自动生成

User click start to take the action to platform. After receiving the trigger, platform give the command to initial the game board. The system with Unity sets game board and shows the map on the platform to show. User swaps items when seeing the game map. The system detects which item changing the position and according to the algorithm to determine how to remove the items. Then the system re-fill the game board. Do the same step above until the game over.

# Implement and Result

## Implementation

According to the game flow, a swap movement activated by the player can result in the following steps of the game: the swap of the neighboring items, the match check around the two swap items, the undo of the swap or the remove of the matched items, the collapse process and the match check for the possible remove, and the new items generation to fill in the blanks with repeated match check, possible remove and collapse, until the new items do not form a match. All of these steps start from the swap of two items, which means the switch of the two column numbers (or row numbers). Therefore, the describe of the whole steps introduced above starts from the definition of the items x and y location.

In the current project, a two dimensional array storing item shapes is used for the swap algorism. The array as well as some useful operations and variables describes the item is encapsulated in a class, referred to as the ShapesArray class.

The following shows a sample piece of the code declaring a two dimensional array of one item, and the dimensions are corresponding to values obtained from the Constants class. In addition, an indexer is also designed to return the specific GameItem through requested row or column.

public class ShapesArray

{

private GameItem[,] shapes = new GameItem[Constants.Rows, Constants.Columns];

public GameItem this[int row, int column]

{

get

{

try

{

return shapes[row, column];

}

catch (Exception ex)

{

throw ex;

}

}

set

{

shapes[row, column] = value;

}

}

After defining the two dimensional array and the GameItem, the swap of the two items (defined in the two GameItems) can be processed by the following Swap algorism. The swap starts from the creation of a backup of the two GameItems, in case that the swap yields no match (or match item number less than three) and the two items need to return back to their original positions. After the backup, the two GameItems swap their positions in the array and, finally, the SwapColumnRow static method is then called in the Shape class, in order to swap the other properties of the two Shape GameItems. A sample piece of code realizing the backup and swap are as follows.

public void Swap(GameItem gi1, GameItem gi2)

{

// The following code is for the backup in case of no match yield

backupGI1 = gi1;

backupGI2 = gi2;

var gi1Shape = gi1.GetComponent<Shape>();

var gi2Shape = gi2.GetComponent<Shape>();

// The following code is to get array indexes

int gi1Row = gi1Shape.Row;

int gi1Column = gi1Shape.Column;

int gi2Row = gi2Shape.Row;

int gi2Column = gi2Shape.Column;

// The following code is to swap items in the array

var tmp = shapes[gi1Row, gi1Column];

shapes[gi1Row, gi1Column] = shapes[gi2Row, gi2Column];

shapes[gi2Row, gi2Column] = tmp;

// The following code is to swap other respective properties

Shape.SwapColumnRow(gi1Shape, gi2Shape);

}

For the undo process, The UndoSwap method is designed to undo the swap via a simple calling of the Swap method on the backup GameItems. A sample piece of undo code is shown as follows.

public void UndoSwap()

{

if (backupGI1 == null || backupGI2 == null)

throw new Exception("Backup is null");

Swap(backupGI1, backupGI2);

}

private GameItem backupGI1;

private GameItem backupGI2;

After the swap attempts from the player’s movement, the next is to check the match, which can be realized through two paths calling the ShapesArray class. One path is the horizontal check whereas the other is the vertical one. In the two checks, the GameItem is treated as a parameter, which contains the row or the column number of the item and any other parameters need to be checked. If the check results find less than three matches, they will return an empty list. A sample piece of the match check code is as follows.

private IEnumerable<GameItem> GetMatchesHorizontally(GameItem go)

{

List<GameItem> matches = new List<GameItem>();

matches.Add(go);

var shape = go.GetComponent<Shape>();

// The following is to check the left items match or not

if (shape.Column != 0)

for (int column = shape.Column - 1; column >= 0; column--)

{

if (shapes[shape.Row, column].GetComponent<Shape>().IsSameType(shape))

{

matches.Add(shapes[shape.Row, column]);

}

else

break;

}

// The following is to check right items match or not

if (shape.Column != Constants.Columns - 1)

for (int column = shape.Column + 1; column < Constants.Columns; column++)

{

if (shapes[shape.Row, column].GetComponent<Shape>().IsSameType(shape))

{

matches.Add(shapes[shape.Row, column]);

}

else

break;

}

// More than three matches are requred

if (matches.Count < Constants.MinimumMatches)

matches.Clear();

return matches.Distinct();

}

private IEnumerable<GameItem> GetMatchesVertically(GameItem go)

{

List<GameItem> matches = new List<GameItem>();

matches.Add(go);

var shape = go.GetComponent<Shape>();

// The following is to check bottom items match or not

(shape.Row != 0)

for (int row = shape.Row - 1; row >= 0; row--)

{

if (shapes[row, shape.Column] != null &&

shapes[row, shape.Column].GetComponent<Shape>().IsSameType(shape))

{

matches.Add(shapes[row, shape.Column]);

}

else

break;

}

// The following is to check top items match or not

if (shape.Row != Constants.Rows - 1)

for (int row = shape.Row + 1; row < Constants.Rows; row++)

{

if (shapes[row, shape.Column] != null &&

shapes[row, shape.Column].GetComponent<Shape>().IsSameType(shape))

{

matches.Add(shapes[row, shape.Column]);

}

else

break;

}

if (matches.Count < Constants.MinimumMatches)

matches.Clear();

return matches.Distinct();

}

If the match check is passed, then the removal function will be activated. The removal can be realized through removing an item from the array (by setting as null). The function will be called once for every match check passed. A sample piece of the removal code is shown as follows.

public void Remove(GameItem item)

{

shapes[item.GetComponent<Shape>().Row, item.GetComponent<Shape>().Column] = null;

}

The last process after the swap, check and removal is the collapse and re-filling in process. After the matched items are removed removal, the remaining items in the related columns will collapse through the method as follows. A search algorism for null items will be activated, and if any null items are found, the nearest top item will be moved to the position of the null item position. The search algorism will continue running until all position of null items are captured by other items and all these null items are stacked on the top positions of the related columns. Then these null items will be replaced by new generated items (i.e. the re-filling in process) containing new parameters randomly. Then the match check, collapse and re-filling in process will be repeated until all the new re-filled items yields no more than three matches.

## Figures of Game Functions

Figure 1 Start Game User Interface Figure 2 Start the first level

图片包含 监视器, 电子产品, 屏幕, 陈列

描述已自动生成 图片包含 监视器, 屏幕截图, 计算机, 屏幕

描述已自动生成

Figure 3 Remove the items Figure 4 Re-filling in Map

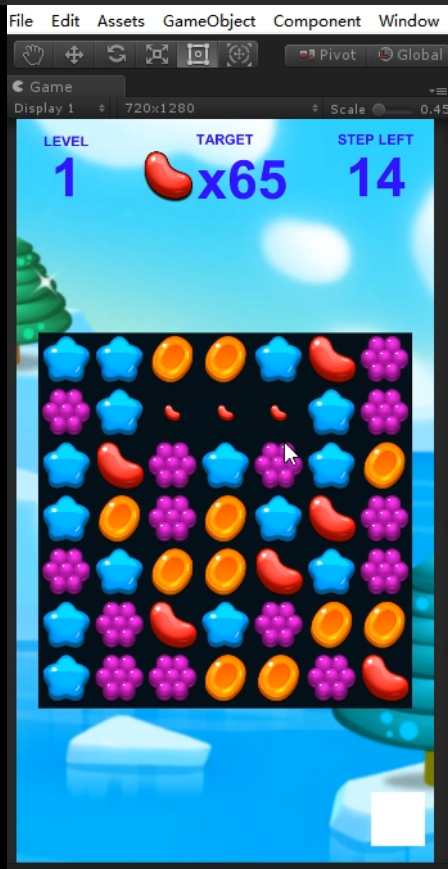
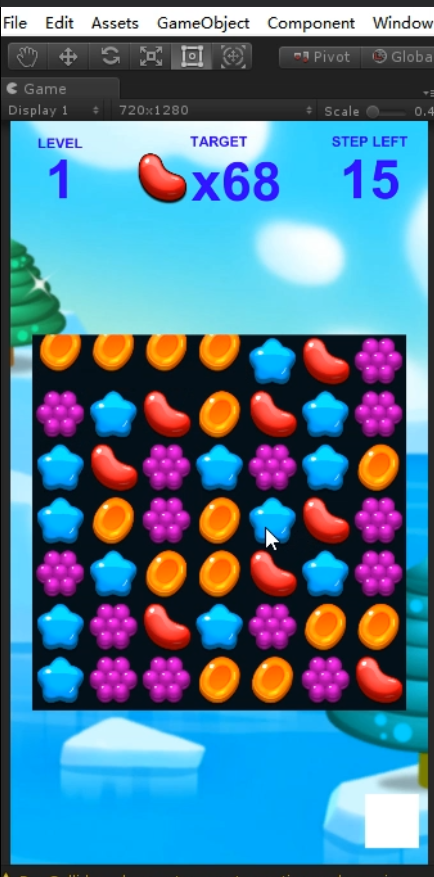
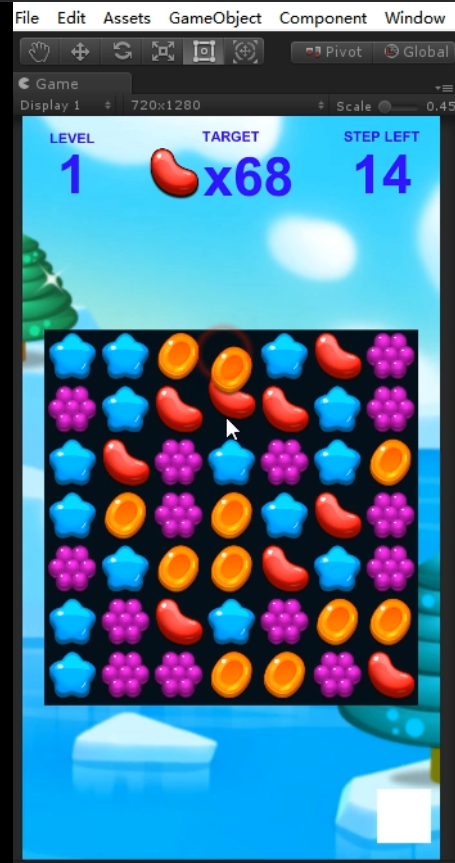
 

Figure 5 Swap Items



# Conclusion

More and more mobile games are enriching our leisure life, but when we think about myself and actually participate in the creation, I find that it is not as simple as I imagined. We are faced with many new challenges.

In the compilation, many good ideas often appear in my mind, but in practice, we find that there are various difficulties. Good works not only test our knowledge reserves, but also test our patience and insight. Therefore, we constantly improve the concept by constantly searching for information and repeatedly modify the work.Although it is very cumbersome, it also brings us a lot of interest and valuable experience. In the actual operation, we constantly explore and solve some problems. Of course, there are still many ideas in the final work that need to continue to practice, and there are still many areas that need to be improved. This requires us to continue our efforts.

Through this graduation project design, we fully integrate the knowledge we have learned in the classroom into practice. It not only solves some practical problems, but also deepens the understanding of professional theoretical knowledge. This provides us with valuable experience in participating in internships and work in the future. At the same time, I understand that learning is a long-term accumulation process. In the future work and life, we should continue to learn and practice, and strive to improve our knowledge and comprehensive quality.

# References

Blackman, S. (2013). Beginning 3D Game Development with Unity 4: All-in-one, multi-platform game development. Apress.

Takahashi, D. (2017). Candy Crush Saga: 2.73 billion downloads in five years and still counting. Available at: https://venturebeat.com/2017/11/17/candy-crush-saga-2-73-billion-downloads-in-five-years-and-still-counting/view-all/

Julkunen, J. (2015). The Future of Match 3 – What You Need to Know – PART II. Available at: <http://www.gamerefinery.com/the-future-of-match-3-what-you-need-to-know-part-ii/>

Lu, G. P., Xue, G. H., and Chen, Z. (2011). Design and implementation of virtual interactive scene based on unity 3D. Advanced Materials Research, 317, pp. 2162-2167.

Norton, T. (2013). Learning C# by developing games with unity 3D. Packt Publishing Ltd.

Smith, P., Hartley, T. P., and Mehdi, Q. H. (2013). C# interpreter and unity 3D for educational programming games. In Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games (CGAMES), 2013 18th International Conference pp. 41-47.