MULTIPLAYER ONLINE FIRST PERSON SHOOTER GAME FOCUSED ON
CREATING STRATEGIC PLAY: SNOWBALL FIGHTER

by

Cong Lin

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Game Science and Design
in the Graduate School of the College of Arts, Media and Design of
Northeastern University

August, 2018
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>1</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>2</td>
</tr>
<tr>
<td>2. BACKGROUND</td>
<td>3</td>
</tr>
<tr>
<td>3. APPROACH/METHODOLOGY</td>
<td>8</td>
</tr>
<tr>
<td>4. RESULTS</td>
<td>20</td>
</tr>
<tr>
<td>5. DISCUSSION</td>
<td>39</td>
</tr>
<tr>
<td>6. CONCLUSION</td>
<td>41</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>42</td>
</tr>
<tr>
<td>APPENDIX</td>
<td>44</td>
</tr>
</tbody>
</table>
ABSTRACT

Real-time strategy games have been a very popular game genre in recent years, and players love to play them with their friends and to stream and share their victories on Social Networking Services. This paper introduces a game project that not only is inspired by the games in this genre but is also based on the activity of traditional snowball fighting. Additionally, this paper discusses how I designed the game to add more flavor into a digital snowball fighting game and how I implemented both the technical and art parts of the game.
1. INTRODUCTION

In my hometown, it never snows in winter. The first time I saw snow in real life was when I became an undergraduate student and went to Beijing. I still remember the moment when I saw the snow floating in the air for the first time, which truly excited me. Furthermore, playing in a snowball fight was an unforgettable experience for a southerner like me. Even though I was already an adult at that time, I still enjoyed throwing snowballs toward my classmates. Hence, when the winter was coming, my friend and I decided to make a game based on the activity of a snowball fight that aimed both to evoke the childhood memory of a player who had had snowball fights before and to convey this type of joy to a player who had not done so. In addition, because it is a digital game, I wanted to add more flavors to the game. In this way, a player who has had a snowball fight before will not only get the same experience of their childhood but also refresh the experience. In the digital world, we can make many new features that the real world cannot produce. In other words, this project focuses on designing an interesting snowball fighting game with features that players will enjoy, regardless of whether they have had a snowball fight in real life. In addition, this project was made by two people, my partner and I. My partner’s name is Zhenhao Xiong, graduated from Entertainment Technology Center of Carnegie Mellon University in 2018. In this paper, if a specific piece of work was implemented by my partner, I would provide a detailed explanation of it. And also in Section 4.4, I listed his work and mine in details.

2. BACKGROUND

2.1 Map Design

PlayerUnknown’s Battlegrounds (PUBG) is a multiplayer online battle royale game that has been very popular since 2017. In this game, up to one hundred players parachute onto an island and scavenge for
weapons and equipment to kill others while avoiding being killed themselves. A particularly important feature is that the map satisfies each type of player, regardless of which strategy he prefers. In PUBG, there are bushes in which players can hide while waiting for other people, and there are buildings that players can explore to find good weapons. Furthermore, there are lots of items that a player can utilize to obtain a competitive edge such as a windows that he can snipe through or a car that he can drive and move fast.

*Overwatch* is also a very popular multiplayer first-person shooter game produced by Blizzard. The difference between Overwatch and PUBG is that in Overwatch, the players are divided into four general categories: Offense, Defense, Tank, and Support. Players on a team work together to secure and defend control points on a map or to escort a payload across the map in a limited amount of time. However, what really attracts me to Overwatch is the map design in this game. This attraction occurs not only because the map is so beautiful (Figure 2-1) but also because it has many different details that players can utilize to obtain a competitive edge. For example, there are roofs and high walls from which snipers can assassinate other players. There are gates and narrow roads for a tank to protect his teammates (Figure 2-2). Furthermore, some players have the ability to change the environment. For example, Mei, a defense hero in Overwatch, has weather-altering devices that can freeze a player and create an ice wall to block the enemy’s path or to help teammates reach higher ground (Figure 2-3).
Figure 2-1 The breathtaking map design in *Overwatch*

Figure 2-2 The tank utilized the narrow road with his shield to protect his teammates from enemy fire
Figure 2-3 The ice wall created by Mei blocks the gate so the enemy need more time to break in.

The virtue of a map is judged by its ability to generate "good" game play, which means that neither side has an unfair advantage and that tactics and strategy must be employed to win (Egenfeldt-Nielsen, 2013). In my game, it would be quite boring if the players stood on flat ground all the time. These two games thus encouraged me to create a map that has different features. Imagine that you have a snowball fight in a much more complex environment, in which you must not only throw snowballs toward other people but also observe the environment and find a better place that suits your strategy. I think that a well-designed complex map will make my game more strategic, which is what I love to do.

2.2 Rules and Mechanics

The rules of a snowball fight are very clear. During the fight, players crouch and knead the snow into a snowball, and then, they throw it toward other people. This construct is simple enough as a basic rule for a game. However, I do not think that it is sufficient for an interesting game. Because in the real life, we can feel a snowball fight in multiple dimension. For example, snow feels cold, sunshine feels warm. And when we hit someone with a snowball, we can see the snowball turn into the snowflakes dispersing in the air and shining in the sun. What’s more, we chase each other, running and escaping from each other,
which makes us tired but excited. Those feelings are stripped when we are making a digital snowball
game. Thus, I need to find or create new mechanics to add more flavors to the game, but at the same time,
I cannot break the basic rule – it should still be a snowball fight.

An idea from PUBG is that we can put different weapons and different armor in the map. When a player
obtains them, the damage caused by a snowball will increase, or the damage that the player takes will
decrease. We were initially excited about this idea because exploring the map and obtaining those
different weapons to win the game seems interesting, which gives you the feeling of exploration and
surprise. But my teammate and I ultimately abandoned it. The reason is that this approach changes the
whole gameplay. If we do so, then players tend to find more powerful items rather than enjoy the
snowball fight, which deviates from our original intention. The game designer should simplify the
approach deliberately to focus the player’s attention on those factors that the designer judges to be
important (Crawford, 1984). Hence, even though this idea does not work, it still makes me clear about
the game – I do not want to make a PUBG-like snowball fight game but a snowball fight game with its
own features.

Another idea, from Overwatch, is that we could endow players with different abilities that support them
and the team in winning the fight (Elias and Garfield, 2012). By and large, it's good for the players to be
able to influence each other, because it makes the game more interesting. However, this approach brings
about another problem – how to prevent a certain type of player from being overpowered. The difficulty
of balancing arises from the emergent complexity of systems. Small tweaks can have unexpected
consequences for a game’s viable strategies and are difficult to reason about (Jaffe, 2012). In other words,
it is difficult for us to create different abilities, and it becomes even more difficult when we attempt to
balance them. However, I still think that it is not a bad idea – in this way, the players will think of how to
use their ability to strengthen their snowball or dodge a snowball from others. Thus, the core is still the snowball fight. In the end, I decided to make a trick to make this idea more acceptable. Instead of giving the ability to the player directly, I transformed them into items that could be picked up off the ground, similar to a boost potion or a power-up. For instance, instead of using an ability called shield protection directly, a player can use this ability when he gets a shield power-up. Thus, everyone has the chance to obtain each ability. Additionally, these abilities have their own duration, so they will not be overpowered as the armor and weapons in PUBG are. Furthermore, because these items will be spawned at specific places, this approach will enhance the conflict among players and make the game much more exciting.

However, in my opinion, this feature is not decisive. It only has a very weak relationship to the snowball fight. As far as I am concerned, the game needs a new feature that is strongly related to the basic rule.

Then, Final Fantasy XIV inspired me.

Final Fantasy XIV is a massively multiplayer online role-playing game published by Square Enix. My idea comes from the new job included in its expansion pack 4.0: Samurai. A samurai uses a sword-like weapon called a katana to deal damage. The rule is that an attack will generate different kenki – a kind of energy. Then, the fighter can release them at the same time. The core idea is that different combinations of kenki will generate different effects, such as dealing more damage or increasing the damage range (Figure 2-4).

![Figure 2-4 Three kinds of kenki - Snow, moon and flower](image)
This gameplay has inspired me greatly. In the game, I need to observe the boss and make a decision. For example, if the boss is moving, using the combo of snow kenki is better than using all three kenki. Although using 3 kenki deals maximum damage, using them at once will take you around 2 seconds to chant and after that the boss is already out of your range. So this is so called raid strategy, which we can utilize in our game. The core idea of a snowball fight is to create a snowball and throw it toward other people. If there are different kinds of snowballs, then we can make snowballs with different effects, which seems much more interesting than always creating the same snowball.

We were trying to use MDA-style to form this rule. Basically, we start from the aesthetics – in this game, one of the aesthetics is strategy. Then we stepped to the dynamics. In the game, we imagined that players would weaken their opponents as well as strengthen themselves in some way. Finally, the last part is to find out the mechanics to achieve what we want. So we designed the Snowball Synthesis System. Through this system, players can make snowballs with different fun effect which may matter the entire fight a lot.

What’s more, during the playtest, we test this design reversely from a player’s point of view. We, as a player, started from learning the mechanics. Then we tried to use this rules. During the playtest, we tried to make different snowballs to understand how they work, then use them towards other players. This was the dynamics part. Once we mastered this rule to some extent, we can use these snowballs weaken other players and get a favorable condition. That gives us a feeling of strategy – the aesthetics we want to convey.

In a word, this idea is the snowball synthesis system in our game and is introduced in detail later.
2.3 Kill and Resurrection

The last aspect confused me for a long time: what happens when someone gets killed? Will the player be resurrected? For this problem, the mechanics of PUBG and Overwatch provide two different solutions. Thinking of a real snowball fight, even if you get hit by a snowball, you can rest and then fight back again. Thus, I decided that the player can be resurrected in a few seconds and join the fight again. Furthermore, I think that it is less comfortable to use ‘kill’ or ‘death’ in such a game. Thus, I changed the death concept to ‘covered by snow’. In other words, if a player is killed, a snowman will spawn in that spot, which is much cuter and more appropriate than a tomb or a dead body in such a game.

3. APPROACH/METHODOLOGY

3.1 Materials

3.1.1 UI design tool – Adobe Photoshop and Illustrator

In this game, I decided to use Adobe Photoshop and Illustrator to design the UI and textures. I also considered other painting tools such as Easy Paint Tool S.A.I (SAI), which is popular among Asian artists. However, painting tools such as SAI are more often used for illustrations with a graphic tablet, and they cannot produce an accurate vector graph. For the UI design, I think that it is better to use a vector drawing to address the different potential screen resolutions. Another important reason is that I am familiar with the tools in PS, such as pen and brush. Thus, using PS and AI will save time compared with that required to learn SAI. Furthermore, because I am not a professional artist, different tools could have less influence on what I draw. The important part on which I should focus is not only how nice the UI looks but also how clearly it can convey information to help players enjoy the game.

Therefore, considering all of the factors above, I choose Adobe Photoshop and Illustrator to design the UI and other textures. The basic idea is that I first use PS to draw a sketch, and I then import it into Illustrator
and make a vector graph based on that sketch. Subsequently, I can export different sizes of the same UI to fit the screen’s resolution.

3.1.2 3D Modeling Tool – Maya
Except for the UI system, most of the art assets in the game are 3D models. Considering the time cost, I decided to use the existing art assets in Unreal Marketplace to create the game scene. However, in the beginning, I still need to build a sample scene to test whether the mechanics work well. Thus, I decide to use Maya to build the sample scene. Because 3D modeling will consume too much time if I want to polish it, a sample scene that consists of basic geometric bodies appears to be a better choice for testing the mechanics and can save time.

3.1.3 Game Building and Visual Effect Tool – Unreal Engine 4
In this project I use Unreal Engine 4 (UE4) as my game building tool. There are two reasons. The first one is that I used it to make games before so I have more experience with it. Games with poor development methodology are likely to run over budget and time estimates, as well as containing a large number of bugs. Planning is very important for every individual and group projects similar (Kanode, 2009). And using a familiar tool helps me a lot to same time. I also considered using Unity at first, but since the games I made in Unity are most 2D games in which the process might be totally different with 3D games, UE4 is a better choice for me in terms of prior skills and risk.

Another reason is that the material editor in UE4 is so strong that I can make many spectacular visual effects. In addition, it is very programmer-friendly. Essentially, the material editor is a form of visual scripting because every node contains a small amount of HLSL code; thus, more programming-savvy users can create their own custom material expression nodes (Vesterinen, 2014). Thus, it is a good choice.
for me to use to create technical art. In addition, good visual effects can make a difference in a player’s gaming experience. Therefore, UE4 is my choice for building the game.

3.1.4 3D Model Package and Animation

Considering the time cost and my ability, I use an existing package to build the character and attach the existing animation to it. These are the only parts in which I use existing assets. Any other materials and textures built in the game are made by myself.

3.2 System Design

3.2.1 Network System based on Steam

In the snowball fighting game, the first task is to make the network work well so multiple players can play together. As shown in Figure 3-1, we use the Steam server as the bridge to exchange information between players. The network process is as follows.
First, when a player creates room for a new fight, he automatically becomes the server. The other players who enter the room are clients. Second, when the game begins, the server player maintains a class called game mode, in which there are many variables, and the functions only run on the server, such as the functions to start the game, end the game and resurrect a dead player. This part was completed by my partner. He connected each player to the steam server. Then, I undertook the following part: focusing on how the gameplay works with the network.

Every player (server or client) maintains his own gameplay data structure, including variables and related functions. The difference is that only the server can change the variables, while clients only copy the result from the server. For example, if a player clicks the left mouse button to throw a snowball, the client will not create a snowball immediately. It will send a message to the server via Steam server, and once the server receives the message, it will send the same message to all of the players (both the server and the clients). Then, the server and all of the clients will create a snowball based on the same information. In fact, the game will not synchronize every move of a snowball because doing so will cost too much time and could result in serious delays. The game synchronizes only the initial state and does not care about how it goes later. To unit the results from different computer, the result only depends on the server data. For example, if a client wants to send a snowball, it will throw it as well as send the initial state to the server. Then the server will notify all the clients to simulate the process by themselves. And if the snowball hits something and deals damage on server, the server will send the outcome to all the client and update their data. So no matter when there is a latency, it could guarantee the outcome is always the same. The only problem is that if a player is in a super bad latency, he may got hit even though he doesn’t see any snowball flying towards him. This is a technology limit, which is ignored in this project. In a word, in this way, the time cost of synchronization will decrease greatly, but each player’s game will see no difference.
3.2.2 Main gameplay system

As shown in Figure 3-2, the main gameplay part consists of the following modules: UI system, Player State, Character, Animation, Battle System, Environment System, and Environment (scene). They work together as follows:

1. **UI**: The UI connects to only the Player State and indicates a character’s variables, such as health, damage, and speed. If we want to change the UI, the game should change the variables in the Player State first. In addition, the UI receives and shows information from the server, which is not shown in this structure.

2. **Player State**: Player State interacts with Character. For example, when interacting from Player State to Character, the game will obtain the variables from Player State, such as the damage and the number of remaining snowballs when a player starts to throw a snowball. When interacting from Character to Player State, the player state will change the value of health when the character is hit by other players.
3. Animation: It is obvious that Animation runs based on the commands from the Character. For example, when the Character runs, the running animation will be called.

4. Battle System: In this game, the snowballs can have different kinds of energy. In addition, the player can use different combinations of snowballs to generate different effects on the enemies. That is a so-called battle system. Basically, the battle system offers lots of functions to support a class called “Character”. And whether a character can use a snowball combo and how he uses it are based on his Player State.

5. Environment System: This system takes the responsibility of building the environment, the weather, the pickable items on the ground and some special events. For example, at a specific time, the system will spawn a pickable item at a specific place, and the player who picks it up will gain a good state which increases the power of a game element such as speed for a while, which is so called gain a buff. A strong buff will help a player win a fight to a great extent.

6. Character and Environment: On the one hand, a Character can change the environment based on a player’s action. On the other hand, the character is influenced by the environment.

7. Visual Effects (VFX): The VFX affects both the character and the environment. For example, the visual effect of a buff could be linked to a character. In addition, the visual effect of an explosion could be related to the environment.

3.3 Character Design
I believe that the artistic style of our game should be low-poly because low-poly models are easier to make and their textures are simple. Thus, using such a model will save a large amount of time.

Additionally, the scene in a low-poly style looks cuter than do scenes rendered in a realistic style. This feeling is very important for a snowball fight game because we hope that the art style will make the game enjoyable and interesting.

The players can choose 3 types of characters. Considering the time cost of finishing a character (model, skin and bones), I decided to use a 3D art asset to create our characters. This approach saved me a large amount of time, which enabled me to concentrate on the gameplay parts. These three avatars look like children, which suits the theme of the game. Because the game idea is to remind their memory of snowball fight at their childhood. Additionally, their colors are simpler than those of some realistic models, so it looks more harmonious to put them in a low-poly world.

Even though the models of the characters are from existing packages, each of the characters still requires a numerical setup to make them different from each other. They also need special animations for player action. Thus, this section introduces the basic numeric setup and animation that are required.
3.3.1 Numerical Setup

The basic rule of this game is that when you hit or make someone a snowman, you can earn points. Snowmanizing a player means a player’s life goes zero and turns into a snowman. It sounds like killing but in a more harmonious way. In addition, when a game ends (it may last for 10~15 min), the player with the highest score wins the game. To encourage players to snowmanize more players, I added a levelling system into the game. Basically, when you snowmanize someone, you level goes up, and your attributes are strengthened, which means that you obtain a more advantageous position. For example, if you have a higher level, you will deal more damage, which makes you more powerful than other players. In this way, I hope that the players will tend to attack proactively and make the fight more exciting. The idea is to avoid an awkward situation - what if all the players hide at some place peacefully? In this way, if you always hide, you can’t win because the one who fights against others more actively will become so powerful that you can defeat. But this doesn’t mean a strong player will be always strong. A dampen mechanism will be introduced later.
The character attributes are established as follows. Each character has three first-level attributes, which are strength, intelligence and dexterity, and four second-level attributes, which are health, damage, attack speed and move speed. Players can affect the first-level values, and the 1st-level values will affect the 2nd-level values, which will ultimately affect the characters.

Players can increase their first-level attributes by leveling up. Each time that one player kills another, his level will increase by 1. In addition, if he is snowmanized, then his level will be half of his previous level, which is so called he gets dampened. This mechanic is both for the strong player and weak player. The strong player knows that he should be careful, too. Because once he becomes a snowman, the punishment is so strong and he will lose most of advantages. This is pressure but also keeps the strong player feel excited. For weak player, this mechanism gives them hope which is that they always have a chance to win. Once they succeed in snowmanize a strong player, they will gain a big advantage at once. So this mechanism will keep them enjoy the game instead of getting upset when they’re weak. What’s more, it guarantees them the feeling of achievement once they defeat a strong player successfully.

The numerical details of a specific character are shown in Table 3-1 and Graph 3-2.

<table>
<thead>
<tr>
<th>Level</th>
<th>Resurrection Time</th>
<th>Strength</th>
<th>Dexterity</th>
<th>Intelligence</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>7</td>
<td>14</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>18</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>11</td>
<td>22</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>
Table 3-1 The numerical setup of knight by level

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>13</td>
<td>26</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>30</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>17</td>
<td>34</td>
<td>17</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>8</td>
<td>19</td>
<td>38</td>
<td>19</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>21</td>
<td>42</td>
<td>21</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>23</td>
<td>46</td>
<td>23</td>
<td>23</td>
<td>23</td>
</tr>
</tbody>
</table>

Graph 3-2 The line graphs of Knight, Ninja and Mage’s attributes by level 1~10. In each graph, one of the lines is dotted line so that it’s easier to see which two lines are overlapped.

As shown in the table, a player with a higher level is much stronger. To balance the power, I also increased the punishment of dying by increasing the rebirth time. Thus, when a high-level dies, he will
not only lose half of his level but also require more time to be resurrected, so other players will have more chances to catch up to his point total.

After playtests, we noticed that this numerical setup might be a little complex, which distracted players from other main features. To avoid this, we hid and removed some of the attributes to make the game strategy much clearer. This will be discussed more in 4.1.2 Character Abilities.

3.3.2 Character Animation

Considering the time cost of making the animations, before creating the game, I designed the minimum amount of animation for a character. The basic animations are as follows:

- Run: A running or walking animation based on the character’s movement speed.
- Crouch: When a player wants to pick up something such as a snowball on the ground, he needs a crouching animation. This also occurs when a player wants to hide behind a low wall.
- Right hand up: This motion occurs when a player attempts to hold the button and aim somewhere.
- Throw: This motion occurs when a player throws a snowball.
- Knead: Players require this animation when they want to make a big snowball from some small snowballs.

3.4 Sample Scene Design

I used Maya to build the sample scene. The sample scene will not be used in the real game. In fact, it is used for developers to test how an environment interacts with the characters. The design concept is as follows.
The sample scene is built in Maya first and then exported in UE4, and the actual model from an aerial view is as follows.

Figure 3-3 The sample scene in aerial view
The aim of each test area is as follows:

- **Low wall testing:** Test the height of a low wall that can protect players from being hit by snowballs when they crouch behind the wall.

- **High and low wall test:** Test whether it is fun when players play near a mixed wall.

- **Slope and upland:** I plan to put essential items on the upland so every player needs to get an item if they want to win. The height of the upland should make it easy for the players to attack another player who is already standing there. Furthermore, it should hide the item so a player cannot see what item he will get until he reaches the upland. The slope should make the player feel that getting to the upland is achievable, i.e., not difficult but not easy.

- **High wall maze testing:** This aspect aims to test how the game will go if players play in a narrow, complex environment. After testing, I found that it is difficult to aim and attack because most of the snowballs hit the wall and nothing happens. Thus, in the game scene, we avoid this type of design.

- **Tube Test:** There are several tubes around the sample scene. This aspect helps a character hide and move to another place secretly.

The sample scene is built to test the game and iterate the process of game development quicker. We have learnt the risks of “building wide”. It means that we don’t build the whole until we know our foundational ideas are sound, otherwise a late discovery of a flaw in the core undermines everything. Therefore, I tried to stripped down the terrain to some more abstract structure and use basic models (cubes, cylinder and tubes etc.) to simulate them. That’s what we called “avoid ‘real art’ as long as possible”.

The game scenes were built by my partner. However, before he finished the scene, I used my sample scene to test all of the gameplay and mechanics. In other words, a sample scene not only saves time compared with that required for a complex world but also greatly helps me test the game.
4. RESULTS

4.1 Game Mechanics

4.1.1 Snowball Synthesis System

At first, I will introduce the design part of the system. As I mentioned in the Background section, to add more flavors to the game, the snowball synthesis system is one of the most important feature in this game. There are three types of basic snowballs: snowballs with ice, fire and dark energy. There is no difference between the basic snowballs, but when the players combine them into a big snowball (see Figure 4.1), it will have a special effect. I use a crystal-like UI to express the energy because the ice has two forms - snow and crystal. The snow form is the basic weapon form as the snowball, so I think the crystal form is a good way to indicate that the ice stores secret energy. Further, storing the energy in ice will make the snow more powerful.

In combination, a snowball with fire energy will increase the damage and burn the enemy, an ice snowball will freeze the enemy, and a dark snowball will increase the damage range and change the environment. Furthermore, the more of the same kind of snowballs a combination has, the stronger the effect that snowball will have.
In this way, there are lots of different combinations that enhance the strategy of the game. The combination I made are as follows.

- **Hell Flame**: Fire * 3. Deal heavy damage to the enemies, and also attach a bad state on the player for 15 sec. The state will deal damage over time which is so called “DoT” (an abbreviation for ‘damage over time’).

- **Fire Trap**: Fire *1 + Dark * 2. Deal medium damage to the player in a wide range and also generate a lava trap on a flat ground. The player who stand inside will get damage over time until the lava disappears or he gets out the range.

- **Snow Storm**: Ice * 1 + Dark * 2. Deal light damage to the player in a wide range and generate a snowy zone on the ground. A player who stays at the zone will be decelerated until he leaves out of it.

- **Ice Storm**: Ice * 2 + Dark * 1. Deal medium damage to the player in a narrow range and attach a heavy debuff to the player. A player with heavy debuff will be decelerated for 10 sec.

- **Eternity Frozen**: Ice * 3. Deal light damage to the player in a narrow range and attach a frozen debuff to the player. A player with frozen debuff can’t move for 5 sec.

The player must master when and how to use different combinations so they can restrict their enemies and strengthen themselves.
The second part is the development part. How does the synthesis system work? In fact, the class PlayerState maintains a queue called BulletCharger. In other words, adding a new snowball to a combination is similar to adding a bullet into a charger. Each time the player adds a new snowball, the charger will first check if it is full. If it is, then the first bullet will be discarded, and the new one will be added into the end of the queue. The blueprint is as follows.

![Figure 4-2 The function of “Add New Snowball Bullet”](image)

Furthermore, once the player hits the synthesis button and shoots, the Charger will pop up all of the snowballs in the queue and create a big snowball based on the synthesis formula.
The function to obtain the type of the big snowball is shown in Figure 4-3. Once the synthesis starts, the charger will calculate the number of each type of basic snowballs. Based on their totals, the function will return a specific type of big snowball. If the combination does not exist, then it will return a big snowball without any damage or effect.

Two of the main functions used in the snowball synthesis system are shown above. Other functions that also help but are less important are not mentioned.

4.1.2 Character Abilities (Pickable Item)

As mentioned in the Background section, the character’s ability is another feature of this game. In the game, every player has potential abilities that can be stimulated under specific conditions. During the
playtest, the feedback showed that it is very difficult for a player to use his ability and synthesize a snowball at the same time. To simplify the gameplay, I hide the ability and add the pickable item to the game scene. In other words, the ability can be used automatically when you pick up a specific boost item in the game. The pickable item is a floating crystal with different particle effects. The design idea of the pickable item is similar to the synthesis system, in which I want to express that the secret energy is stored in ice crystals and that it will support you in this snowball fight. Because both snow and ice crystals are forms of ice, this approach makes the design pattern of the game unified.

For the crystal, different particle effects have different effects on the player: SpeedUp, Shield, Sneak, Heal50%, DamageUp and RangeUp.

- SpeedUp: The crystal is shown in Figure 4-4. When a player gets it, the character’s speed will be twice his original speed for 15 sec. This helps the player dodge other players’ attacks.

![Figure 4-4 SpeedUp PowerUp](image)

- Shield: The crystal is shown in Figure 4-6. It will generate a 10-sec shield around the player. In other words, for a 10-sec duration, the player will not take any damage from others.
The shield is shown in Figure 4-6-1. Additionally, when a snowball hits a shield, the shield will have a special effect that shows that the energy of the explosion is diffused over the whole shield, as shown in Figure 4-6-2. How to create this VFX will be introduced later in the section on VFX.
Sneak: The crystal is shown in Figure 4-7 and is covered by thick fog. The sneaky power-up will make the players transparent, which will help the player launch a sneak attack.

The process of sneaking is shown in Figure 4-7-1. It will take the player 2 seconds to become completely transparent. Furthermore, to prevent this from being too overpowered, even though a player becomes
completely transparent, the refraction of the player will not change, so other players will still see him vaguely (Figure 4-7-2). How to make this VFX will also be introduced in the section on VFX.

- Heal50%, DamageUp and SpeedUp: These three types of snowball are shown in Figure 4-8-1. Unlike the powerups above, these powerups do not need any special VFX and simply increase the specific attributes of the character. However, to make it clear which buff a player has, the same particle effect will be shown on the player’s body, as shown in Figure 4-8-2. Thus, other players will know which crystal he has obtained based on the particle effect around his body.
4.1.3 Damage Calculating

In a fighting game, damage calculation is a very important part of the game process. There are three main factors that influence the damage dealt in the game: damage range, damage type and damage cause.

- **Damage Range:** At first, the snowball deals damage only when it hits a player. However, after the playtest, I found that it is too difficult to aim at a player accurately if he keeps moving, which makes the player feel very frustrated. Learning works best when new challenges are pleasantly frustrating, i.e., when learners feel that they are at the outer edge of but still within their ‘regime of competence’. In other
words, these challenges feel difficult but ‘doable’. Furthermore, learners feel – and obtain evidence – that their effort is paying off in the sense that they can see, even when they fail, how and if they are making progress (Gee, 2005). To solve this problem, I decided to make all of the snowballs deal damage in a specific range. The range of the damage is calculated as follows (Figure 4-9). When a snowball hits somewhere, it will apply a radial damage to all of the characters in the range. Additionally, the number of damages will decrease from a maximum to 0 based on the distance between the player and the explosion.

![Figure 4-9 Radial Damage](image-url)

- Damage Type: There are three damage types in this game: physical damage, magical damage, and DoT (damage over time). At the beginning of this project, I assumed that a character might increase his physical/magic defense to decrease the damage that he takes. However, I later found that this approach is too complicated for this game. Thus, I simplified the design and removed the attribute of defense. I retained these three damage types in the case of future development, but right now, the physical damage works the same as the magical damage.
- Damage Causer: Snowball Fighter is a real-time strategy game, which means that the players should know how much damage he deals, how much damage he takes and from whom he takes the damage. Thus, the damage causer is a very important record that tells the player the information about the damage. As shown in Figure 4-10, if someone takes damage, the server will broadcast a message to inform the player. This becomes much more important when you need to broadcast a kill record, such as a double- or triple-kill. A double-kill means you kills two players continuously during a short time, which means you’re highly skilled and gives you a feeling of achievement. This will be introduced more in part 4.1.5.

![Image: Got 17.945 Damage From Cong.](image)

**Figure 4-11 Damage Record**

After playtests, I noticed that level system and power-up system have similar effect in some way. For example, when a player’s level is up, his attribute of attack is increased, which works similar to the condition when he gets a damage-boost power-up. These two systems distracted players a lot because they didn’t know which they should pay more attention to, killing or searching for the power-up. To reduce this distraction, I combined the level system with the power-up system and removed some of the attributes. On one way, a player can get a level-up only when he gets a power-up. Killing a person doesn’t guarantee a level-up any more. On the other way, most of the attributes has been removed except attack and movement speed. This aims to unify the level system and the power-up system because there is no power-up that can increase the attributes except attack and movement speed. Unifying these two systems makes the goal much clearer – fight for power-ups to get stronger.


4.1.4 Pickable Item Generator
The first question is why do we add a pickable item into the game? We have learnt a good-looking interest curve of the game. It contains basic loop with higher and higher climax. For our game, the basic loop is to observe the environment, find enemies and make snowballs. Then we started to think about the climax. A climax might happen if two players met. But what if they all hide? To avoid this, we design the power-up system. Basically, it will provide the player powerful effect – some are permanent and some are temporary. If a player always hides, he will miss these power-ups while other players will become too strong to defeat. In a word, through this system, our idea is to add more conflicts between players and avoid the silence for too long which is awkward.

The second question is how to implement this feature? In the game, I used two blueprints to generate a pickable item randomly: Power-Up Controller and Power-Up Spawner. The power-up spawner simply provides the controller with its position. The process of generating the power-up can be divided into three procedures.

1. Obtain the position to generate a power-up: The controller will search all of the existing positions of the spawner first. Then, it will select one of them randomly as the position of the next power-up. The blueprint is shown as follows.

![Figure 4-12-1 Randomly pick one position from all](image)

2. Spawn a new power-up: After the controller knows the spawn position, it will start to spawn a new power-up on it. The type is random, but it can be ensured in debug mode. The blueprint is as follows.
3. Loop: The spawn process is a loop. Each time the controller generates a new power-up, it will fall asleep for a period of time (e.g., 10–15 sec randomly). Then, it will wake and start the next loop of generating a new power-up. To achieve this step, I set a timer for the loop, which is shown as follows.

4.1.5 Kill and Reborn

As mentioned in 4.1.3, the damage causer is recorded in the game. In this way, we know who makes the last valid attack when someone is killed. Hence, every time that a player kills someone, the game will
compare the new log with the old killing history. If he kills more than one player over a short time, then the server will broadcast a kill message to encourage that player (e.g., double-kill, triple-kill). I think that this is a very effective way to enhance the tension or gameplay atmosphere. For example, if you have watched the League of Legends World Championship Series, you will know that situation where you are overwhelmed by the deafening applause and screaming when a player gets pentakill.

Once a player gets killed, he cannot perform any action until he is resurrected. The time-cost of the rebirth is shown in Table 3-1. Additionally, he will lose half of his current level, which is the so-called “death punishment” and could have more of an impact when the level is higher. However, if the attackers wait near the rebirth position to kill the same person repeatedly, which results in a very bad experience, I set up many rebirth positions and give the player a 5-sec shield when he is resurrected. Thus, for the attacker, the position where the killed player will appear is random, and he is protected by a shield when he is resurrected. The code is shown as follows.

![Figure 4-13 Respawn a killed player (Only run on server)](image)

**4.2 UI design**

In the game, there are two parts of the UI: the static UI and dynamic UI. A static UI does not mean that it will not change but that it is fixed on the same part of the screen. Additionally, it presents detailed information about the player and the game because players always prefer to have relevant and sufficient information, which allows them to interact meaningfully with the game mechanics and the gameworld.
(Llanos, 2011). Furthermore, a static UI means that a player cannot see any other player’s static UI. In contrast, a dynamic UI floats above the player’s model and can be seen by other players.

4.2.2 Static UI

The static UI is shown as follows in Figure 4-14.

![Figure 4-14 Static UI](image)

Area 1 is the damage record. It will show information about the damage that you have taken from other players. Additionally, if a server has broadcast an important message, it is shown in this area.

Area 2 is a buff bar. Each time that a player obtains a debuff (such as frozen or burning DoT), it will be shown as an icon with a number on the bar. In Figure 4-14, there is no number under the buff icon, which means that the duration of this buff is infinite. The buff icons are downloads from Google Image so I can focus on making the buff system work well.
Area 3 shows the ranking of all players in the game. Each time that you deal damage to someone or kill someone, you will earn a certain score. Additionally, when the game ends, the player with the highest score is the winner. It is acceptable to use a key to hide/show the ranking, but I want to decrease the use of different keys to make the game more friendly for beginners. Thus, I think that the ranking should be fixed on the screen.

Area 4 is the snowball synthesis system. Information about this system was presented in section 4.1.1.

Area 5 shows the basic information on a character. It updates the attributes of a character from the Player State.

4.2.2 Dynamic UI (Floating UI)

The dynamic UI is floating above the player and can be seen by other players. It presents only the basic information about a player, such as his name, health and level. And it also shows what buff a player has. This part works similar to the part in the static UI. A dynamic UI is shown in Figure 4-15-1.

![Figure 4-15-1 Dynamic UI](image)
The most important feature of the dynamic UI that I implemented is that it should be hidden when a player is out of another player’s sight. For example, when a player hides behind a wall or stays a sufficient distance from a player, this UI should disappear; otherwise, it seems like cheating.

To achieve this, I set a line trace between the dynamic UI and every player. If the line is blocked by something, then the UI will hide itself. Additionally, the script will calculate the distance between two players. If it is out of range, then both UIs will be hidden. Thus, if a person hides behind a wall, he cannot be found by his dynamic UI. The blueprint is shown in Figure 4-15-2, and the result is shown in Figure 4-15-3.

Figure 4-15-2 Hide his dynamic UI when a player is invisible

Figure 4-15-3 Hide his dynamic UI when a player is behind a wall
4.3 Art Assets

4.3.1 Visual Effect

Visual Effect (VFX) plays a very important role in the game. This section will introduce the implementation of three VFXs: the shield, sneak and particle effects of the pickable items.

- **Shield**: The VFX is shown in Figure 4-16-1. We can divide this VFX into three parts. The first part is the main body: it is a spherical shield with honeycomb textures moving on it, which gives a sense of technology. The second part is its edge. As is shown in Figure 4-16-1, an edge intersects the ground and an edge in the air (the edge of the circle) that has a higher brightness. The last part is the on-hit VFX. Once a snowball explodes on the shield, it will send a position into the shield’s blueprint. Then the shield will generate a VFX on that position, which is a shining circular honeycomb texture that diffuses from the position over the whole shield. This VFX is attempting to express that the damage is shared across and weakened by the shield. This type of dynamic VFX can leave a deep impression on the players. A preview and the setting are shown in Figures 4-16-2 and 4-16-3.

![Figure 4-16-1 VFX of the shield (Left: Normal, Right: Being attacked)](image)
- Sneak: The preview and setting of the material are shown in Figures 4-17-1 and 4-17-2. I generated the texture used in this material using the cloud function in Photoshop. This VFX is called dissolve VFX. The degree of how a material dissolve is controlled by a variable is called the dissolve
amount. In the blueprint, I use a timeline to change this variable from 0 to 3 to achieve the process of becoming transparent.

Figure 4-16-1 Preview of the dissolve VFX

Figure 4-16-2 Material setting of the dissolve VFX

- Particle effects of the pickable item: I use the particle editor of UE4 to create the particle effect (Figure 4-17-1). The shapes of the particles are made by me using Illustrator. The shapes include healing, sword, shield, snowflake and up-arrow (Figure 4-17-2).
4.3.2 Character Model, Animation and Sound

The character model is from a 3D model package in the UE4 marketplace. The animation is downloaded from [https://www.mixamo.com/](https://www.mixamo.com/). The sound that we use is from the default package of UE4.

4.4 Division of the Project

This section is to clarify the work of my teammate and mine.

My teammate worked on:
1) The network structure: including the logic how to create a server/client, how to find a existing server and how a client connects to a server. And the user interface related to network is also included, such as create a new server, finding and joining an existing room.

2) Building the real game scene: we bought the scene model such as rocks and trees from Unreal Store. And my partner use those things to build the game scene.

3) Character Animation: Neither of us have the skill of creating animation. So we bought the character animation assets and my partner works on attaching those animation onto the models.

4) Character Selection User Interface: My partner creates the UI that you can select the character you like as your avatar.

I mainly work on the gameplay, the list is as follows:

1) Based on the network structure built by my partner, I completed the real logic about how to send message from client to server and how to synchronize the data among all the clients.

2) The User Interface of the game play - including all the elements when you’re in a snowball fight. There are two parts of UI, the static UI such as player’s health points, attack, speed, buff bars and the UI of the snowball synthesis system. Another one is the dynamic UI, such as other players’ health points, level and their buffs. Also I fulfill the feature that if you hide behind something or far enough away from another player, your dynamic UI will be hidden in case other players find you.

3) The Player State Class - saves all the data a character has and synchronize the player state from the server to all the clients. What’s more, I wrote the related function of updating the data, for example, a function to calculate the damage and your rest health points when you get hit.

4) Pick-able Item System: including how to generate a pick-able item at a certain place, and I implemented all kinds of pick-able items, both the function and the visual effects. Take the
pick-able item that can heal you for an example, I implemented the function that once you get it, you will heal 50% hp for 5 seconds as well as I created the visual effect that you will be surrounded by the green cross particles while being healed. The shield VFX is my proudest work, which I have explained earlier.

5) Snowball Synthesis System: including its UI, related functions to implement this feature, the synthesis formula of all kinds of big snowballs and their unique visual effect. For example, I draw the user interface that fill the synthesis system with two dark energy and one icy energy. And then based on the formula, this kind of big snowball will create an area that slows your speed. The visual effect is that, in this area, a lot of snowflakes are flying, dancing and spinning in the air, which is made by me.

6) The buff system: including the user interface and its function logic. For example, a burning debuff will be shown on the top of the static user interface and your dynamic UI. The debuff will deals damage over time on your character. If you get another burning buff before it disappears, its duration will be refreshed instead of showing two same debuffs.

7) Environment System: including the sample scene and the snowy particles. The details of the sample scene has been explained above. The snowy particles is to simulate a real snowy weather, which is supported by the graphic card.

5. DISCUSSION

For game development, the most important part that I learned is the entire process of coding an online game using UE4. In this project, I learned how to synchronize the player states among the players. Furthermore, I came to understand the mechanics of how a server communicates with clients. It is very important that you clarify which functions should only run on the server and which ones should run on
both the server and a client. Another important thing that I learned from this project is how to make technical art assets. I must say that as a programmer, it is difficult for me to draw beautiful pictures. However, technical art can be generated by code and simple textures. Even if I am not extremely good at art, this approach has given me another way to generate spectacular VFX. For example, the shield VFX is my favorite among all of the technical art assets. In the beginning, although this VFX was very impressive, I could not find any tutorial for making it on the internet. There were a few videos on YouTube that showed the effect, but they did not provide any hints on making it. Thus, to make this VFX, I watched their video repeatedly, analyzed the VFX and divided it into some simpler parts; then, I tried to finish the small parts one by one. As mentioned above, I divided the shield VFX into 3 parts, achieved them one by one and combined them. Finally, I obtained the shield VFX according to my expectation, which I am still very proud of to this day.

For game design part, the first thing I learned is to protect your core idea. That means, think twice when you want to get some inspiration from other successful game. This happens when I tried to get some good features from PubG. Those features coming from other game may not suit your game, or even worse, they may change the core experience of your game. That would be horrible when you’re trying to build your own game.

The second one is not to build too wide, especially at the beginning of the game development. It’s easy for a designer to be overwhelmed by tons of good ideas when a new project starts. Usually, we are excited of all the ideas and don’t want to abandon any of them. But that’s very dangerous. Time and energy are limited, so we have to grab the most important cores of the gameplay and make all the sub-features support the core. What’s more, this make the whole idea of the game united. Otherwise the player will be confused - what is the main thing I should do for this game?
The third thing is the interest curve. For most cases, it’s not difficult to build a game with a linear interest. That’s the basic loop of the game. But it’s not enough if we take all the time to revise it. The feature that really make your game impressive and stand out of other game is the one which creates the climax in the game. For example, the basic loop of my game is throwing the snowball. But the real fun part are the time when players fight for the pick-able item. The conflicts among players puch the game to higher and higher climax, which happens a lot during the game. Therefore, we should both think of the basic game play as well as the features that can create climax while designing the game.

The last thing is to listen to your players. Their advice and feelings matter a lot. For a game designer, the most upset thing is that all the things of your game are good, except it’s not good to be fun. Therefore, never be satisfied with your ideas if none of no-designers played it before. One of my professor told me a thing which impressed me a lot: you must feel pity after you choose to be a game designer, because once you’re a designer, you’re not a normal player anymore. You already have bias, whatever you notice or not. So the only thing to help you is to listen to your players. This helps me a lot while designing the game - when I have a new idea which seems so exciting, it always helps me keep calm - it’s nothing unless it pass the test of the players.

My game still has some limitations, and there are some features that could make the game better, but I did not ultimately implement them due to a lack of time. First, there are many combinations for the snowball synthesis system. However, I only made three different combinations because a new combination means a new texture, a new effect and new VFXs, which would have required too much time. Thus, in this game, I simply picked three types of combinations that I think are classic and understandable. Second, a lag or stuck will occur occasionally due to the network, and sometimes, the synchronization will fail, or some
important information will be lost. If I want to optimize the network, I will spend too much time on it. Hence, I had to put this issue aside, which is a pity for an online game. Third, there is no clear guide for a beginner to watch. Thus, when people play the game for the first time, it could confuse them for a while. Providing guidance for an online game is a very complex and independent system. I hope that I can finish this feature in the future. The last concern is the appearance of the dead bodies. Earlier, I said that to make the deaths cuter, a snowman would spawn at a player’s position when he died. I think that this idea is a very smart move, and I want to implement it in the future.

6. CONCLUSION

Throughout these two semesters, I transferred the snowball fight from an idea into an online game. For the design part, I searched many popular games and obtained a great deal of inspiration from them. Even though only a few of these ideas are used in the game, I learned how to make choices among many good ideas to fit the core of a game. Furthermore, I made many numerical setups to fit the game and tuned the players’ powers based on the feedback from the playtests, which gave me very valuable experience about the numerical setup.

For the programing part, I understood how the network works among the server and clients and how the server and clients communicate with one another. Additionally, I built the entire gameplay system by myself, including the battle system, UI system, buff system, snowball synthesis system, powerup, character, and environment.
For the art part, I learned a substantial amount about technical art. Except for the character model and their animations, which came from an existing package, I created all of the UIs, textures and VFXs that I needed in the game. For the 3D models, I created the sample scene in Maya and exported it into UE4 to test the gameplay. For the UI, I designed the entire UI system, including the static UI and dynamic UI. Thus, a player can see detailed information about himself as well as simplified information about other players. For the VFX, I used the particle system of UE4 to build the snowy weather and the particle effects on the powerup and characters. Furthermore, I used a material editor to create the shield VFX and dissolve VFX. Both of them look very impressive.

Although there are limitations in the game, I think that the gameplay fits the idea of a snowball fighting game. The game is derived from a real snowball fight, but it is more than a snowball fight. The real world and the digital world are so different that you will lose lots of things while you’re trying to translate a physical game into a video game. But video game also has lots of potentials because it only depends on your imagination. That’s why this unreal world are so charming that attracts both players and designers into it. Imagination plus some design rules, an old physical game will reborn in a new form.
REFERENCES


APPENDIX

Snowball Fighter Design Document

1. Game Objectives
   a. Players can earn the points by hitting other players with snowball.
   b. The game lasts up to 10 min. The one who earn 500 pts first or the one who has most points is the winner.

2. Game Rules
   a. Each player has 100 hp (health points), 20 basic damage.
   b. The player can gather snowballs on the ground. The number of snowballs a player can carry is up to 10.
   c. The basic damage of a snowball is 20 and the damage range is 2 meters. The damage will decrease linearly from the center of the explosion to its edge.
   d. A big snowball can be synthesized by 3 snowballs and will has special effects.
   e. The pickable powerups will be generated randomly in the map. They will strengthen the player in a short time.
   f. Once a person gets killed, he will get resurrected after 5 sec.

3. Mechanics
   a. Controls
      W/A/S/D: Move up/left/down/right
      Space: Jump
      Z: Collect small snowballs
Left mouse button: Hold it to aim at someone, release it to throw the snowball

1/2/3: Fill the charger with the fire/ice/dark energy

E: Synthesize the big snowball

Ctrl: Crouch

b. **Snowball Synthesis Formula**
   
i. Fire + Fire + Fire: Deal 30 damage at once and attach a burning debuff to the player. The player with burning debuff will get 5 damage every 3 sec.

   ii. Ice + Ice + Ice: Attach a frozen debuff to the player. The player with this debuff can’t move in 5 sec.

   iii. Fire + dark + dark: Generate a lava area when hitting the ground. When a player walks in this area, he will burn and get 5 damage every 3 sec.

c. **Powerups**
   
i. Damage/Speed boost: Double the damage/speed of a player

   ii. Heal: Restore 50 health in 3 sec.

   iii. Shield: Protect a player from any attack in 10 sec.

   iv. Sneak: Make a player invisible for 15 sec.

4. **Game Scene**

   The scene with different area is to test different functions in the game. In the snowball fight, a good place means a player can hide and dodge the attack easily. So I design this sample scene to test how the places with different features work.
The goals of different features are as follows:

1. **Low wall testing**: Test the height of a low wall that can protect players from being hit by snowball when they crouch behind the wall.

2. **High and low wall test**: Test if it’s fun when players play near mixed walls.

3. **Slope and upland testing**: Put powerups on the upland and encourage players to fight for it.

4. **High wall maze testing**: Test how the game goes if players play in a narrow complex environment.

5. **Tube Test**: There are several tubes around the whole sample scene. It helps characters to hide and move to another place secretly.