Dragonborn: A single hero multiplayer game

Angelos Kyriakopoulos
University of Amsterdam
University of Crete
January of 2014
Bachelor thesis - Computer science
angelkyriako@gmail.com

ABSTRACT

Creating a video game can be a complicated matter. Many different fields such from hardware and software engineering to graphical art, music and writing, are combined together to achieve what a lot of people can enjoy as a game. This work is presenting a game created by an individual developer with little resources, to emphasize on the fact that nowadays game development is more accessible to people other than the global well established studios. Dragonborn is a 3d single hero multiplayer online battle arena (MOBA) game with elements of real time strategy and role playing games. Each player controls a baby dragon (thus dragonborn), that is progressing through leveling up during the game. The dragon can be trained in different manners based on a stat system and can use certain skills to battle with each other. The skills are developing further by leveling up, as the game proceeds, until their maximum potential is achieved. Players are separated in teams and need to co-operate with their allies in order to defeat the opposing teams. Dragonborn connects users over the network with a room based system, meaning that multiple short length games can be played, with different players each time. The game was implemented (for pc) using the unity3d game engine and the photon unity networking plug in (PUN). In the following sections, we are going to see further all that has been stated above in terms of gameplay and implementation. In the end we will conclude by discussing the strengths and weaknesses of the game and improvements that would bring the game at a whole new level.

Categories and Subject Descriptors

D.1.5 [**Programming Techniques**]: Object-oriented Programming.
D.2.3 [**Software Engineering**]: Coding Tools and Techniques – *Object-oriented programming, structured programming, top-down programming*.
D.2.4 [**Software Engineering**]: Software/Program Verification – *Assertion checkers, class invariants*.
K.8.0 [**Personal Computing**]: General – *Games*.

Keywords

Dragonborn, game, dragon, arena, multiplayer, lobby, room system, stats, skills, spells, leveling, real time strategy, RTS, role playing game, RPG, Unity, Photon, Daikon Forge GUI.

1. INTRODUCTION

Dragonborn is a game that targets mostly teenagers that like playing RPG or strategy games. It involves a lot of thinking, as a skilled player has to think of the best strategy and make the correct choices in order to win the game. However, the controls of the game are fairly simple and users can also enjoy it or even win by being carefree and acting randomly, without too much thinking. Dragonborn was based on aspects that can be found on [1] and [2] that are games which share same genre as well. The design of the game and the choices regarding the gameplay are based on the lenses described in [3]. It is implemented with the Unity3d game engine [4], Photon network engine [5] and Daikon Forge GUI middleware [6] among other assets from the unity asset store [7]. In addition to that, helpful resources were used or studied from the unify wiki [8], C sharp design patterns [9], hack & slash RPG tutorials [10] and stormtek's RTS guide [11]. By executing the game, users can enter a valid username that will let them transfer immediately to the lobby screen. The graphical user interface will provide information about the available rooms that have already been created by other users. The user has the option to either join in an existing game or create a new one and wait for other people to join. The person that creates a room is able to modify a few game options that can change the rules of the game, according to their own or the other joined user's preferences. The creator of a room is going to be referred from now on as the Host and all the other joined users as the Joined players. A user is transferred to the room screen by entering in a room. There they can pick their player slot and team among ten different player slots and ten different teams. Furthermore, the GUI informs the players about the game preferences and the ongoing state of the teams. The Host is the master of the game and thus the game preferences can only be modified by their client. He is also the only one who can start the game, but only if every Joined player has stated that they are ready to proceed. The users can communicate with each other via a chat window to agree on the teams they will form and the game preferences. In case one is not happy with the state of the game, based on the other player's option, it is possible to leave the room and go back to the lobby screen to enter another room. In addition to that, when needed Hosts are able to force other users from the room they created by kicking them. As soon as every player is ready the Host can start the game and transfer everyone to the main game.

A dragonborn is going to spawn for every player in the room, each with a different color based on the player slot picked previously by the users. The baby dragons are as strong as their numbers in each different stat (strength, stamina, agility, intelligence, charisma). During the game the players are rewarded with a number of training points every time they level up, that they can use to manually raise the stats they prefer. The stats define the combat power of the dragon such as its health, speed and attack. Other than that, they determine which skills can be used by the player as the game goes on. A player is able to use a maximum of four skills while fighting. Those can be picked from a skill book and each of them is explicitly connected to their owner stat points. Even though the skill options are limited at first, more skills are unlocked as the player levels up and spends training points to their stats. The skills that are available for use are based on the dragon's stats and thus the raising choices a player made during the game. Therefore the dragon's combat

usage can be a lot different, depending on the strategy a player chooses to follow.

The starting area is an invisible terrain that terrain is trying to resemble a sky area which is used as a waiting point for the players, but also during the gameplay. From now on we will refer to that area as the Heaven. When in Heaven the dragons cannot fight with each other and therefore, they need to transfer to the actual terrain, which during the next paragraphs will be called the Earth. This transfer can happen by entering into a portal that will teleport the player characters to the Earth, when they are ready for battle. During battle when a dragon's health goes to zero it counts as a death for that player. Usually that death is caused by an effect of an opponent's skill. In this case, that opponent is rewarded with a kill point and it is possible for their whole team to gain experience points from it. The winning conditions of the game are based on the kills or deaths each team has achieved as a whole. When any of the teams meets the winning condition, its members have won the game and it is immediately terminated. The users are able to see the statistic results of the game and transfer themselves to back to the lobby screen where they can join another new game. Thus dragonborn offers endless replayability. For the rest of the article when the term game shows up it could not only mean the application as whole but also a gaming cycle that started, played and has finished just like when playing a game of chess or backgammon.

2. LOBBY & ROOMS

2.1 Meeting at the lobby

Dragonborn does not include any single player mode. Games can only be played between human players that are connected over the internet with the game's room system. Because of that each game played can progress in a different manner depending on the choices the players make each time. The lobby, as depicted in figure 1, is the place where players can meet and connect with each other to form a game. Either by creating a new game and waiting for other players to join, by joining directly to an already existing game or by letting the system choose automatically a room for them, players can achieve that connection.

Figure 1. The lobby.

2.2 Inside the room

When connected, players are transferred to the room of the game. This is the place where the rules of the game will be defined, by the players themselves. The main logic behind that is that, since the each game is played for a limited amount of time until the winning requirements are met, a system is needed to let the players experience different gameplay features each time. This way, even if the main features of the stay the same, the game itself seems to be less tiring as more and more game cycles are being played. Therefore, players are free to choose different team formations, difficulty or winning conditions among other preferences that will be described further below. As mentioned above the creator of the room is the one who is in charge and there are certain rules that only the host player can set. Figure 2 below depicts the room's graphical interface.

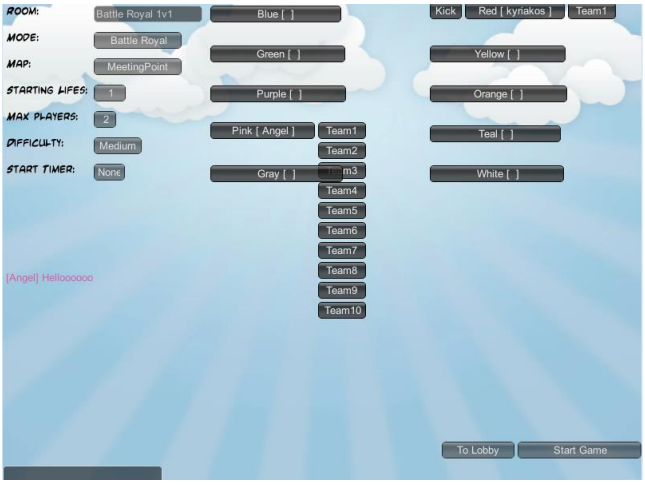

Figure 1. The room.

The rules that players are able to modify are described on the subsections below.

2.2.1 Mode

The mode of the game defines the winning conditions. Those are relevant to the kills and deaths of each player during the game. There are only two modes on the demo, the Conquerors and the Battle Royal.

2.2.1.1 Conquerors

In this mode the team that wins is the one that will reach a target number of kills first. The kill count of each team is calculated by summing up the kill counts of the individual players that are part of it. In conquerors the host can pick the target kills count that each team must reach in order to win, among a specific list of numbers.

2.2.1.2 Battle Royal

In this mode the team that survives win. Players have a limited amount of lives they afford to lose. The host picks the starting lives count for all the players. During the game players that lose all their health, die, respawn and fight again. However in this mode when a player dies as many times as the lives count they do not respawn again and can only observe the "living players" play. When the only "living players" left, are part of the same team, the game has reached its end and that team wins.

2.2.2 Map

The map is technically the terrain that the fighting takes place. A terrain can be just an open area, when another can be full of obstacles. The length and the terrain objects may differ between maps. There are only two different stages available (Forest, Plain) to demonstrate this feature on the demo.

2.2.3 Maximum players

This is the number of the maximum players that can be connected in the room. It can be between *two* and *ten* and it is displayed next to the title of the game on the rooms panel of the lobby screen.

2.2.4 Difficulty

The difficulty can change significantly the gameplay. Different difficulty can change the time length of the game or give a little push to beginners that are not experienced with playing the game. Features like leveling, the after death waiting time or the visibility radius around the player character. Those changes can be described further on the next sections.

2.2.5 Starting timer

This is nothing more than the time that the players have to wait before teleporting to the battle area after the game starts. It can give a chance to players to figure out what strategy they could use depending on the formation of the teams and generally pick their starting stats and skills.

2.2.6 Title

This is the text that is going to be displayed on the available rooms panel. The first impression one gets before entering a game is tightly relevant with the title. Therefore, meaningful game titles can help players understand of what preferences are favored by the host. For example a title "2v2 battle royal beginners" passes the basic information to players that are seeing the room successfully. Instead the title "Join asap" is not descriptive at all. The first game is probably going to be of easy difficulty and on battle royal mode and the host is looking for 3 more players to form two opposing teams. However, we have no idea what the host has in mind for the second game.

2.2.7 Teaming up

Even though the host is the one that can change the variables above, players can change their team (among ten different teams) by themselves. Since there are ten different teams it would be too annoying for the host to have to change all the player's teams on their own. That way joined players can state which team they prefer as soon as they join a room. However, further discussion regarding the team formation or the other preferences of the game is possible through the game chat if needed.

2.2.8 The chat

Sometimes the players may need to discuss a little more about the preferences of the game, even though the host is in charge of it. Different people want different things and the game has to give at least the chance for them to find a common solution among the teams and the game preferences. Of course no one can ensure that everyone will agree on the same rules but then that can happen in every game that includes teamwork. There is always the choice of leaving a room and join or create another one. Eventually everyone can find what they are looking to play.

3. GAME ENVIRONMENT

3.1 The dragonborn hero

Dragonborn is a game that is oriented based on the player characters. For that reason we need an analytical description of what a player can do with their hero. In other words, how the hero can be controlled in order to move and act during the game. Also, in what way players can interact with each other and how a player can know about the existence of other players on the terrain.

3.1.1 Controlling the hero

The game can be played with the mouse and keyboard of a PC. Users can move their hero over the terrain, by right clicking with the mouse on a target location. The speed of the movement is highly relevant to the agility stat of the character that we will discuss in the next section. The baby dragon can use skills in order to act during the game. Skills could either cause damage to enemy dragons or have supportive effects to allied dragons or self. Each hero has exactly four skill slots they can fill in with skills that are available from the skill book of the game. Each slot is represented by one of the "Q", "W", "E" or "R" keys. When one of those buttons is pressed, the corresponding skill will be either enter on a targeting mode or directly casted based on what type of skill it is. Skills that are directly casted are positioned automatically on the world space based on the caster's location over the terrain. On the other hand, targeting skills are used in two steps. By pressing one of the skills buttons the skill on that skill slot is being selected and entering the targeting mode. Then depending on the mouse location, the user can choose the destination of the skill and by pressing again the same button or the left mouse click the skill can be launched. However, if the user right clicks or presses another skill button the selection of the previous skill will be undone. From now on we are going to call the first type of skills basic skills and the second type targeting skills.

3.1.2 Player's view

Until now we have in mind a basic idea of what a user can see when playing the game. However, we have be more specific in order to completely comprehend a player's view. First of all, the world is being displayed in a top down manner that remind a real time strategy game. This viewport was chosen, because of the targeting skills that exist in game. Since a player has to target locations on the terrain in order to choose the destination of certain skills, the RTS view is the most suited perspective in order to keep the complexity of the game on a low level. Players have already to think a lot about their stats and skills combinations. Adding complex mechanics on top of that, could be tiring or annoying for them. The terrain has a fixed height and every player character and skill objects are also positioned on a predefined height. This way player have to think only about two dimensions when targeting somewhere on the field. Because of that limitation the RTS camera seemed the most suitable, since players cannot easily notice the fixed height of the game objects. On the other hand, the same limitation would seem unnatural from a first or third person camera viewport. Except of the terrain (with the static objects that exist on it) and the baby dragons, there is a need of graphic user interface to provide all the necessary information needed. In figure 3 are depicted the terrain of the *meeting point* main stage among with the GUI windows and the two dragonborns that are controlled by the players. The GUI windows one can interact with in game are:

- The terrain map window that pinpoints the locations of the dragons. Players can set the window visible or invisible by pressing the "M" key.
- The game info window that displays the kills count, deaths count and the team of each player in game.
- The game preferences window that displays the rules applied for a particular game. Players can set the window visible or invisible by pressing the "G.
- The character window that provides analytical information about the hero's stats and attributes. This is the window that lets players to raise their stat values at the cost of training points. Players can set the window visible or invisible by pressing the "C" key on the keyboard.
- The skill book window that displays the list of all the skills (title, description, mana cost, cooldown, availability and requirements), that a player can choose to fill their skill slots. Players can set the window visible or invisible by pressing the "K" key on the keyboard.
- The action bar that displays the skill slots of the player, the health, mana and experience bars. The action bar is constantly visible while the fight is still in progress.
- The game statistics that display general information about the game once it is finished. It is displayed when the game ends and is an extended version of the game info window. Players can set the window visible or invisible by pressing the "G".

Figure 3. The Earth & GUI.

3.1.3 Dragonborn's vision

Players are playing the game in teams and compete with each other, trying to have their team win the game. Players depending on the team they are part of, can experience differently the game. In general dragonborns that are part of different teams are not visible to each other from distance. Each dragon has a certain vision radius that represent how far they can see and understand the presence of their opponents. The base vision radius length is based on the difficulty of the game, meaning that in a game of easy difficulty players can see farther than in a game of medium or hard difficulty. In addition to that, the vision radius length is also relevant to the intelligence and charisma stats of the character. However, even if each dragon has their own vision radius, allied dragons share the same vision, meaning that not only they can see each other, but they also see the opponents that are inside the radius of their allies. This feature enhances the teamwork among allied players, since they can assist and help each other easier when needed. Furthermore, the map window on the top left of the screen that displays each allied or visible enemy hero's position over the terrain. The map can help a lot when playing a game on a large scale terrain, since the viewport of the camera seem limited, when we consider the length a terrain could have.

3.1.4 Dragonborn's bonds

The goal of the game for all the players is tightly connected to the health of each dragon. During combat each player will have detailed information about their current health visible at the action bar. However, the experience and difficulty of the game would be different, depending on the information players know about their allies' or enemies' health or not. Dragonborns share bonds in a way that they can approximately "sense" the life force of each other. However, the information they can learn about others through those bonds, are not as detailed as the information about their own health. Actually, there are three different health states that dragons can sense about other dragons and the players are informed about that by a colored sphere above the characters:

- **Healthy state**: The current health is at least 50% of the maximum health. During this state the sphere is not visible at all.
- **Dangerous state**: The current health is between 25% and 50%. During this state the sphere is colored yellow.
- **Emergency state**: The current health is below 25%. During this state the sphere is colored red.

This enhances the teamwork between them. By knowing the health state of their allies the team members can assist each other when needed. By default the dragons are not bonded with opponent dragons. However, not being able to see their attacking target health state may raise the difficulty of the game. This is why the bonding system is also based on the difficulty of the game:

- **Easy**: All dragons share bonds with each other. Therefore all players can see other players' health states.
- **Medium**: Only allied dragons share bonds with each other.
- **Hard**: There are no bonds and therefore players have to pay attention on the skills' visual effects that are used during combat, to try to understand how damaged other players are. However, the visual effects are poor on certain skills and because of that, the bonding system is treated just like the medium difficulty.

3.2 Stats

3.2.1 Main stats

There are five main stats that represent the combat power of the baby dragon. Those stats can be upgraded by the players with training points that are acquired by leveling up. The maximum training points that can be acquired are sixty (when the maximum level is reached) and by spending one training point, a stat can be raised by one point. The training points players get at each level

are predefined for each level and can be between one and five. In addition to that, there is a limited amount of upgrades that can take place. Actually, players can upgrade their stats as many times as their current level, meaning that in the long term they can only upgrade them as many times as the maximum level of the game. An upgrade is considered the action of raising by one, one or more of the main stats of the character.

There are several reasons why this method of stat upgrading was picked. First of all, a stat cannot get greater than a predefined threshold (it is the level in our case, but any other threshold could be used), but this feature could easily be applied even if players could have an unlimited number of upgrades. The main reason is that the players that want to build their stats in a way that they would have one or two maxed out stats at the maximum level, are forced to do this gradually as they level. For example, they cannot have a stat with value twenty five when they are still at level twenty, even if they will have been rewarded with more than forty training points by then. They need to keep some of their training points unused on the early levels, when players that choose to have a more distributed stat build, will have all their stat points used at most of their levels. (@NOTE: see later if I want to add more details here.) In the following subsections more details regarding the main stats are provided.

3.2.1.1 Strength

The character's physical capabilities. It mostly focuses on the physical damage the dragon can deal and the physical resistance to opponents' physical attacks. However, this stat also boosts its health. If one wants to deal and withstand a great deal of physical damage this is what they should focus on raising.

3.2.1.2 Stamina

The vitality of the character is represented mainly by this stat. The character will be able to withstand enemy physical and special attacks with ease and will regenerate quicker over time.

3.2.1.3 Agility

The speed of the character's actions is based on this stat. An agile dragon can not only move faster or use its skills more often but also has a good chance to evade opponent attacks or to deal critical damage when attacking.

3.2.1.4 Intelligence

The character's intelligence boosts the critical chance and the visibility of the dragonborn. In addition to that, its special capabilities are taking a boost both on attack and defence. One could say that is the exact opposite of the *strength* stat.

3.2.1.5 Charisma

This is the ability of having a leading role that provides support to the team. It slightly boosts the vitality and the vision of the dragon but this is not it stands out. In fact it is the only stat that can boost healing skills of a dragonborn that can be of great importance in a fight.

3.2.2 Stat distribution

The previous section was very abstract. The exact use of each stat during gameplay is not clear yet. In this section we are going to introduce the attributes of a baby dragon. Actually, the attributes are used to measure the combat use and power of each player character. The main stats are only the values that group different attributes together in order to reduce the complexity of the game. It is a lot harder for the player modify fourteen attributes than five basic stats. Players may feel overwhelmed of the paths they could take when upgrading their dragon's stats and this attribute grouping takes care of that issue. Furthermore, it is fairly easy to scale up the attributes by raising their number and make the combat system of the game more realistic, without necessarily adding more stats. This way players are not forced to learn about those changes immediately. They can still play the game as they are used to, since the only values that they modify are the stat values that are the same. Nevertheless, the attributes are displayed in the game for players that may seek more information about them. They could check how attributes are updated when a stat is raised and find out exactly each stat's distribution to attributes. In the following table depicts the increment of the attribute values for each stat point.

Table 1. Main stat distribution over the attributes

	STR	STA	AGI	INT	CHA
Health	8	30			5
Mana		35		12	8
Physical power	10				
Special power				6	
Physical defense	4	8			
Special defense		6		3	
Health regeneration	.03	.08			
Mana regeneration		.12			.04
Movement bonus			.021		
Attack speed bonus			.016		
Critical chance			.005	.01	
Evasion chance			.013		
Vision bonus				.015	.06
Leadership					.7

There are two type of attributes the vital and the normal attributes. The first are described by a maximum value and a current value when the second only by their maximum value. When a stat is raised by one point, the maximum value of the attributes of the group it represents is raised as well as much as the corresponding value on the table above.

3.2.3 Vital attributes

The game itself is based a lot on the vitals of the player characters. The current value of those attributes is modified very often by the skills, players cast on each other. The current value of a vital is the one that represents the quantity of it and the game actually takes into account to trigger certain in game events (damage, healing, player death, skill casting .etc).

- **Health**: points that represents the life of the character. When the current value goes to zero the character is considered dead. When a character receives physical damage or is getting healed the current value of the health vital is modified accordingly.
- **Mana**: points that represents the energy of the character. An amount of mana is needed in order to cast skills. When players do not have the sufficient points

needed for a skill they cannot cast it. When a character casts a skill, receives special damage or is getting healed the current value of the mana vital is modified accordingly.

3.2.4 Normal attributes

- **Physical power**: points that boosts the physical damage when using an attack skill.
- **Special power**: points that boosts the special damage when using an attack skill.
- **Physical defense**: points that reduces the physical damage when being hit by an attack skill.
- **Special defense**: points that reduces the special damage when being hit by an attack skill.

  Physical power or defense is relevant with the modifications of the health vital of a character. Likewise special attributes are relevant with the modifications of the mana vital.
- **Health regeneration**: the points that are added to the current value of the health vital per second, during the game.
- **Mana regeneration**: the points that are added to the current value of the mana vital per second, during the game.
- **Movement speed**: percentage that boosts or reduces the base movement speed of the character over the terrain.
- **Attack speed**: percentage that reduces or boosts the time it takes for a skill to be used after being cast (cooldown).
- **Critical chance**: the chance of dealing double damage on enemies when using attack skills
- **Evasion chance**: the chance of ignoring the received damage of the opponents' attack skills
- **Vision radius**: percentage that boosts or reduces the vision radius of the character.
- **Leadership**: points that boost or reduces either the physical or special healing effects of a support skill.

3.3 Skills & effects

Now that we know more about our dragonborn hero, it is time to look into the skills of the game. In general, by using skills, one can harm or help themselves or other players. Actually, a skill is a group of effects. When a skill "hits" a dragon, its effects affect the hero. When a dragon is affected by an effect their attributes' values can get modified. The maximum value of normal attributes and the maximum and current value of vital attributes can be modified by effects. An effect is activated at the moment a character is affected by it and after a certain amount of time (depending on the effect's type and variables) it eventually gets deactivated. During the activation and the deactivation procedures certain modifications take place based on the effect type and variables. In the following subsection we will take a closer look on all of the effect types that can be part of a skill.

3.3.1 Effects

First of all, we need to separate the effects with respect to the timing of the activation and deactivation procedures. If the question is "when", there are three different effect types:

- **Basic**: Those effects are activated once. Basic effects can be used to represent a permanent effect like direct damage or healing. For that reason, there is no need for a basic effect to be deactivated.
- **Lasting**: These effects are also activated once. However, they can live for a certain amount of time. After enough time has passed, they eventually get deactivated
- **Over time**: An extended version of the lasting effect. The addition here is that over time effects have a certain frequency value. Based on that value, the effect gets activated multiple times until it gets deactivated.

To continue, we have to ask "how" each of the effects affect a character. To answer this question the effects are separated to the following types:

- **Vital modifiers**:
  - *Damage*: reduces the health's current value.
  - *Health heal*: raises the health's current value.
  - *Mana burn*: reduces the mana's current value.
  - *Mana heal*: raises the mana's current value.
- **Maximum value modifiers**:
  - *Buff*: raises the maximum value of an attribute.
  - *Debuff*: reduces the maximum value of an attribute.
- **Special**:

  Effects that do not modify attribute values and are specific enough to not be able to be categorized with other effects fall into this category.
  - *Stun*: the character cannot move or cast skills.
  - *Silence*: the character cannot cast skills.
  - *Invulnerability*: the character cannot be affected by damage or mana burn effects.
  - *Push*: the character is moved slightly to a certain direction.

Example 1: Let us consider an offensive skill composed of the following effects:

1. Basic damage effect.
2. Over time mana burn effect that lasts for 4 seconds and has frequency of 1 second.
3. Movement speed debuff that lasts for 4 seconds.
4. Over time physical power debuff that lasts for 3 seconds and has frequency of 1 second.
5. Stun that lasts for 2 seconds.

During the activation of the skill above, a dragonborn, will be affected by the effects that is part of that skill. The timing of the activation and deactivation procedures based on each type is described by the table below.

Table 2. Activation and deactivation of effects

Effect/ Second	0 sec	1 sec	2 sec	3 sec	4 sec
1	activate				
2	activate	activate	activate	activate	Deactivate
3	activate				Deactivate
4	activate	activate	activate	deactivate	
5	activate		deactivate		

For the first effect no deactivation is needed since the damage is a permanent effect. On the other hand, even if the mana burn is also a permanent effect, it is activated over time. Therefore the second effect needs to be deactivated when the time it lasts has passed. The same is happening with the fourth effect. However, in this case, during the deactivation procedure, the physical power's maximum value will return back to normal. The third and fifth effects are activated only once just like the first one. However, they are lasting effects and when the time comes they will be deactivated. The stun effect will be undone and the movement speed will return back to normal.

3.3.2 Skills

We have already discussed that a skill is practically a group effects. However, before continuing we need to describe two features that are irrelevant with those effects. To begin with, there is a limitation regarding how often a dragonborn can use a skill. A cooldown time is part of each skill and when one casts it, the player has to wait for that time to pass in order to use this skill again. This cooldown time is reduced based on the attack speed percentage, meaning that agile dragons can use skills more often than slower ones. Another skill feature completely irrelevant with the cooldown time is the requirements of the skills. The main idea is that even though every skill is technically available for every player, each skill comes with certain minimum and maximum requirements that need to be fulfilled in order for a player to use it. Those requirements are based on the stats a player chooses to upgrade. For example a skill could has the following requirements:

- Intelligence must be greater than five (min).
- Strength must be greater than seven (min).
- Agility must be lower than fifteen (max).
- Strength must be lower than eighteen (max).

The requirements exist to differentiate the skills players can use based on their stat build. Even though there are no different roles that player could choose from the beginning for their dragonborn, this feature gives the sensation of a role playing game just like if it was. Players with different stat builds will unlock different skills as they are progressing during a game. In addition to that, since many games are being played, players could choose to upgrade their dragon's stats in a different way each time. This feature can keep player interested in the game for significantly more game hours than if the skills were standard for everyone. Furthermore, because skills can be picked and stats can be upgraded anytime during a game, players might change their starting strategies depending on how the battle in proceeding at any time. This can be proved to be challenging though, since helpful information one needs to have in mind regarding their opponents' stats or skills, are not available for display to everyone. Players, need to guess their opponents and teammates stats based on the skills they can use (since skills have stat requirements) and how effective they are on other player characters. Finally, the maximum requirements are useful also for balancing related matters. We can restrict the users to not be able to use skills that could be overpowered under certain circumstances. For example, a skill could be perfectly balanced in general but when one stats to have more than twenty points on their strength stat. With a maximum requirement of eighteen on the strength stat this skill will be balanced and the game would be fairer. Of course this is not the best way one could tackle a game balancing problem. However, it is really attractive since balancing issues can be fixed with a lot less effort. After all, nowadays games are still being updated after their release and this is a quick way of resolving those matters, before providing a proper solution (that takes a lot more time) regarding an imbalanced feature.

It is time to further discuss about the skills one could find in game, based on the skill system that was constructed. Grouping effects into a skill is one way of adding variety to the skills that exist in game. However this is not enough, since there other features that need to be represented through different skill types. As we have already mentioned at [3.1.1] with respect to the way a player can cast one a skill can be either basic or targeting. The behavior of the first is based on the position and rotation of the caster character. The second takes into account the position on the terrain the player is targeting and the position of the caster character. However, a skill can be also categorized based on other factors too. Before proceeding any further, we need to define the life cycle of a skill:

1. ***Casting***: The procedure needed for the player to use a certain skill.
2. ***Behavior***: The movement of the skill object once it is created and placed on the terrain and the triggering conditions of it.
3. ***Triggering***: The events that can take place when a skill is activated. During this phase the effects of the skill will be activated on certain dragonborns.
4. ***Destruction***: The skill cycle ends with the skills destruction, just after it is triggered.

If we consider the behavior of a skill, a skill can be:

- **Directly activated** meaning that is instantly triggered either at the location, the player has chosen (targeted skill) or at the point where the dragon is located during that time (basic skill).
- **Projectile** that moves from the dragon's location towards the destination the player has chosen (targeted skill) or towards the direction the dragon is facing (basic skill). Projectiles come with certain speed and range values. Their movement speed is based on the speed value. These skills are triggered, either once their distance from their origin is greater that their range or when they hit another dragon. Projectiles can be triggered by allied dragons, enemy dragons or both.

- **Trap**, a basic skill that is placed at the position of the dragon that casts it. Trap skills come with a time to live value. They are triggered when their time to live has expired or when an enemy dragon triggers it by walking over it.

The characters (allied, enemies or both) that could trigger a skill is a different matter than which characters can be affected by its effects. For example, certain skills are triggered on collision with enemies. Those skills would certainly affect enemy characters damaging them, but could also affect the allies of the caster in order to support and help them. We have already mentioned that projectiles and directly casted skills can by triggered by any dragonborn and that trap skills can be triggered only by enemy dragon characters. It is time to see how a skill can affect dragons that are either allies or enemies with the caster. During the previous section we described the skills as a group of effects. However, this was too straightforward. Actually, a skill is composed of the following groups of effects:

- **Enemy effects**: are attached to dragonborns that are not part of the caster's team.
- **Ally effects**: affect characters that are part of the caster's team.
- **Passive effects**: affect only the caster.

When a skill is activated on any character, the system will affect them with the corresponding group of effects, based on their team and the team of the skill's caster. This skill structure helps a lot with the construction of a large variety of skills. Furthermore, balancing the skills and therefore the game, is a problem that is already broken into pieces that can be solved easier now.

To continue with the skills description, we need to introduce the area of effect or AoE skills. They come with a radius value and based on that value and the distance of the dragons from the activation point of the skill, they can affect multiple targets. Any skill type of the ones described above can be either AoE skills or not. Identifying the characters that are affected by the skill, is a procedure that takes place during the triggering phase of the skill cycle. An AoE skill has a limit on how many dragons it can affect at once. When enough characters are affected, it will not affect more which is a limitation enhances the variety of the game. Let us consider a weak damage AoE skill that has a maximum character limit of six characters and an almost two times stronger one that can affect maximum two characters. Even though the second is limited to affect only a few characters, it could be more efficient than the first one, when less than four dragons are gathered. On the other hand, when more than four players are gathered, using the weaker skill could prove to be wiser, since the summary of the damage achieved would significantly greater.

3.3.3 Skill escalation

We have analyzed the skills and the effects and how they are structured in the game. However, there is another important aspect of those that needs to be examined. A skill needs to be scalable in a way that as a player is improved by leveling up, their skills are getting also improved. For that reason a level requirement is part of every effect. During the triggering of a skill only the effects that are qualified will be activated. In other words, if the level requirement of an effect is greater than the level of the caster that uses the skill, the effect is ignored. As the character is leveling up less effects are ignored and therefore the skill gets more powerful. In addition to that the mana cost of the skill is also raised as more effects are used when it is triggered. This is happening because the mana cost is an effect feature and not a skill feature. Therefore, the more the number of the *valid* effects of a skill, the greater the mana cost that is needed in order to cast it. As the game is progressing a player has to "pay" more to use skills with high level effects. We need to remember that the stats of the dragons are also raised as they level up, meaning that their mana points can be increased, making the use of "pricey" skills feasible. However, not all stats can raise the mana points of a dragonborn. For that reason, not all of the skills come with high level effects attached. Those are more attractive to players that have chosen a stat build that does not provide that much mana points. Now we are starting to see through the different roles a dragonborn can have during battle. While a character that has mostly raised its intelligence and stamina stats could withstand more damage and cast high level skills, another could raise its strength and agility. This would allow them to hit more often and with more power, even though their skills would mostly be composed of lower level effects, since high level skill could prove pricey for them. However, the second character could have a teammate that has a high charisma stat. This player could heal their mana points, allowing them to use skills that cost many mana points more often. Teamwork is essential for a team to win the game. Different strategies can be applied in each game by a team, based on the stat builds and the skills that are used by its members. This is actually one of the most appealing aspects, if not the most appealing one, of playing dragonborn.

3.4 Combat events

3.4.1 Player death

When a dragonborn's health reaches zero the character is considered dead, the deaths counter is incremented by one and the player loses a life. On death, the hero will teleport to Heaven (figure 4) and will have to wait for a certain amount of time [4.2.1.6] to restore its health and mana in order to be allowed to teleport back to the field. In battle royal mode, when players run out of lifes, they are not allowed to teleport back to the field and therefore can only watch the progress of the game. For that reason all dragons are becoming visible and by pressing the the "T" key, one can switch the target of its camera. With this feature the player can have a better view on how the game is progressing.

Figure 4. The Heaven.

3.4.2 Player kill

When a death takes place, the player that was the owner of the effect that delivered the final blow is considered the killer. When that is not an ally a player kill is achieved and their skill counter is raised by one. Since a player kill is not achieved when a dragon dies by an allied player\s effect and there are skills in game that cause damage to team members, players could take advantage of that. However, this feature could irritate many users and for that reason when one dies, they lose an amount of five, ten or twenty percent of their experience points based on the game difficulty.

3.4.3 Experience gain

Dragons may gain experience when any of their team members achieve a player kill. All players that are part of the same team as the killer and are near the location the kill took place, will gain experience based on the level of the defeated dragon. The radius that defines which players are near or not is also based on the game difficulty.

3.4.4 Leveling up

Players need to make them stronger by killing their opponents and gaining experience. When its experience bar is filled a dragon will level up and require a greater amount of experience points in order to reach the next level. By leveling up a player is awarded by training points that can be used to upgrade the stats of their dragon and therefore its attributes.

4. IMPLEMENTATION

4.1 Technology

4.1.1 Unity

The game was implemented using the unity3d game engine. It is an engine built on top of the .NET framework and therefore it supports different platforms. One could use Unity to assemble art and assets into scenes and environments, add lighting, audio, special effects, physics and animation. The programming languages that are available for use are C#, Unityscript which is an extended version of javascript and Boo. This project is entirely created in C#, since it includes a stricter syntax that sometimes could prevent certain bugs. Because unityscript is a dynamic typed language, it has so many shortcuts that are invitations to make mistakes. Instead, the syntax of C# is closer to Java and C++ (popular languages for game development), meaning that many popular software patterns of those languages could be used with not many modifications also in C#. Furthermore, using different languages for one project, even if it is supported by Unity could prove to be problematic. This is mainly the reason C# was chosen for the development of dragonborn. Unity is a component based game engine, which means that one has to think in terms of a “Part of” architecture rather than an “is A” one which is more inheritance based. This was the most difficult and challenging part of the development of the project, since it is not an easy task for one to think in a modular way when being used to inheritance based solutions. To conclude, Unity provides a class that is called *Monobehavior* that every script component that is part of a game object, has to derive from. Classes that inherit from *Monobehavior* can use the events that come with the Unity API.

4.1.2 Photon Networking

Photon Networking is a network engine that provides a plug-in for Unity (PUN). It provides an integrated room system through its API that was really helpful during the implementation of the dragonborn room system. In addition to that it provides a cloud that can host the server of the game and can support a maximum number of twenty concurrent users. This feature accelerated the development process, since managing a self-hosted server is a task that takes time from other development operations. Photon does not allow custom server logic for the cloud, but despite that the solution that runs on that was sufficient for the implementation of the game. Through its API, PUN provides mainly two ways of achieving networking communication. The first is via remote procedure calls (RPC), Users are able to send messages to other players telling them to execute the RPC that is needed. RPCs are used mainly for game events or actions that do not take place often. For example the room GUI or the updating of the kill and death counters are operations that are synchronized only with RPCs. When objects need to be observed often in order to synchronize their state between clients, a serialization method can be used. This method runs ten times per second and through a stream, clients can read (receive) or write (send) information to other clients. An observed game object will write data to the stream so that all other clients can read those data. This feature is used to synchronize the position and rotation of the baby dragon or other moving objects, since those variables are being continuously modified during the game. The cloud is running a non-authorative server, uses a peer-to-peer networking logic with the two communication methods described above and thus dragonborn does the same. The PUN plug in extends the *Monobehavior* of Unity with the *Photon.Monobehavior* which includes a networking related API.

4.1.3 Unity Asset store & Third party solutions

Most of the art that was used for the game was downloaded from the unity asset store, where artists or developers can upload and sell or give away their assets. Except from models, animations, audio files or utility scripts people (either big companies or independent persons) can create and upload their own tools. Daikon forge GUI is a really useful library that extends the Unity editor and was used for the user interface of the battle stage. At first everything GUI related like the lobby or the room was created with the native library that comes with unity. Since there was no need for high frames per second count at that part of the game, it seemed to be working fine at the moment. When GUI windows started being created for the main stage of the game the FPS count started to get really low. The Daikon forge GUI library was chosen mainly to reduce to the minimum the draw calls of the GUI module and therefore raise the FPS of the game. Despite that, after getting familiar with it, creating different windows to provide the information needed to the players of the game was a lot easier that with the native Unity library. Finally, the unify wiki is a place one can find a lot of information and help regarding unity related matters. Many well-known software patters, coded based on the unity game engine structure or other unity specific works can be found there.

4.2 Main components

In this section we are going to describe the main components of the dragonborn implementation. We will discuss about the components of the dragon game object, since anything in the game is focused on that hero. In addition to that, insight is going to be given regarding how an effect can actually affect a dragonborn. We will continue with the skill system

implementation and conclude with the managers that contain concentrated functionality regarding different aspects of the game.

4.2.1 Player character components

As we have already mentioned the player character is the center of the game and everything is oriented around it. Like any game object in a Unity project, the hero includes different components.

4.2.1.1 Model

The model includes all the information regarding the logic state of hero. A list of the skills that the player has chosen and uses from their action bar is part of this component. The values of the stats and attributes, level, experience points, training points, kill and death counter of the dragon are also included here. The skills' cooldown timers updating and the regeneration of the health and mana of the dragon take place on every frame rate in this component. In addition to that, the script *listens* to input events from the player input singleton regarding the use of skills by the player. In general procedures like experience gain, level up, raise the kills or deaths count, skill usage and stats and attribute modifications are part of the model.

4.2.1.2 Movement controller

This component is related with the transposition and the rotation of the hero over the terrain. The movement of the dragon is based on its base movement speed and the movement speed bonus attribute that is part of the model component. The movement controller also *listens* to input events from the player input and moves accordingly the player character game object. Furthermore, it is responsible of animating the character between different states by taking into consideration its current movement speed.

4.2.1.3 Vision controller

The visibility feature is achieved by dividing the game world into different layers and controlling which player characters are on which layers by this component. A dragon can be part of one of the layers below in each client:

- **Allies**: This layer is visible to the user. The hero that they control among with all the dragons that are in the same team are part of this layer.
- **Visible Enemies**: Also visible to the user, dragonborns part of enemy teams that are located near any allied dragon exist in this layer.
- **Hidden Enemies**: Invisible to the user, player characters that far away from allied team members are part of this layer.

When a game starts the layer of each character is being chosen based on the team formations. All the allies are in the *Allies* layer during the whole game. However, enemies are moved between the other two layers. The vision controller includes the functionality of moving player characters between those layers. If any allied vision controller finds out about an invisible enemy that is inside the vision radius of the character, it moves the enemy to the *visible enemies* layer and vice versa. To achieve that, when the game starts all the vision controllers of enemy players are disabled in each client, based on the local player character's team. Therefore, only the local or any allied remote player character can change the layer of the enemy characters. This is a procedure that runs locally on each client.

4.2.1.4 Life sphere controller

Based on the current health and team of each character and the difficulty of the game a life sphere may appear on top of a dragonborn. This component is responsible to turn the sphere from visible to invisible or change its color based on those factors.

4.2.1.5 Color picker

Each dragon has a different color in order to be differ with the others. The color picker is used when the game is starting. After all players have chosen their colored player slot, it will set the corresponding texture for each baby dragon based on that slot.

4.2.1.6 Respawn controller

When a dragonborn's health goes to zero it is considered dead. Then it is teleported to *Heaven* and has to wait for a certain amount of time (respawn timer). In general the respawn controller is inactive during most of the game. However, when a player dies it is activated and sets the respawn timer based on the level, kill and death counter of the player character and the difficulty of the game:

- **Easy**: level/2 + level * ((kills /2) / (deaths * 2)) seconds
- **Medium**: level + level * (kills / deaths) seconds
- **Hard**: level*2 + level * ((kills *2) / (deaths / 2)) seconds

In other words, the easier the difficulty, the more the penalty is reduced. In addition to that, the easy difficulty provides help by punishing more, players that have little deaths (skilled) over players that die more often (beginners). Furthermore, we can see that the medium difficulty is more neutral and when in hard difficulty, players are punished even more when they have achieves more kills. One could doubt the idea of punishing good players. However this feature really improves the chances of a team that was losing to turn the table (by keeping high skilled players off the battle for longer) and thus improves the element of surprise.

4.2.1.7 Network controller

This controller is responsible for synchronizing the dragonborns between clients. While all the other components of the player character object run locally, this component if the one that "observes" it through the network. The position and rotation among with the attributes of the dragon are all the variables that need synchronization through serialization. However, for certain battle events like player kills or deaths that do not need continuous synchronization, remote procedures are called in clients. RPCs take place only in the scope of the network controller, which is a policy that was used for every game object during the implementation. In addition to that, only one networking component is allowed per game object in order to maintain easier the networking functionality of the game.

4.2.1.8 Effects attached

The effects that we discussed at [3.3.1] are also components that are attached onto the player characters when a skill is activated on them. The previous components are all static in a way that they are part of the dragonborn game object and stay there for the whole duration of the game. However the effects stay attached for a finite amount of time on a dragon game object. Basic effects

are destroyed instantly the frame that were attached, when lasting and overtime effects stay attached until they get deactivated. The effect components modify the values of the attribute variables that exist in the model component of the player character to simulate all the effects that were described at the effects section.

4.2.2 Managers

Another important module of the implementation is the managers module. This module includes a group of singleton classes that manage different aspects of the game. Some on them derive from Monobehavior or Photon.Monobehavior since they use the Unity or Photon Unity API, while others are used as variable or constants holders. Managers that inherit from one of those classes need to be components of a game object. However, more than one of those components may have networking functionality. This game object is the only case that a game object has more than one network controllers, since its only use is to hold those singletons and nothing else. In the following subsections we are going to describe the most important of those managers.

4.2.2.1 Game manager

The game manager is the most important manager of the project and requires both the Unity and the Photon API. First of all it is used as a *pool* of every player character object in game. This makes the look up for any dragon more efficient, since the build in solution Unity provides to find a game object in a scene is usually slow. When a player joins a room a request is being sent by the game manager to the host's machine in order to synchronize the pool state. After the response from the host each client updates the pool of player characters locally every time another player joins or leaves the room. In addition to that, the game manager includes the functionality for initializing or restarting a game. In other words it is responsible for creating and destroying objects when scenes are being loaded. Finally, checking the winning conditions of the game after a player kill or death, broadcasting system messages in different clients based on game events and maintaining all information regarding the user of each dragon (name, color, team) are tasks that also part of the game manager.

4.2.2.2 Combat manager

Includes battle related procedures and information and inherits the functionality of the *Photon.Monobehavior*. The experience radius length is accessed from this manager and the procedures of every combat event [3.4] are also part of it. In addition to that, the combat manager is used to provide information regarding if players are allies or enemies. Finally, it is also used to instantiate any object that is created during battle (skill projectile, aoe effect, level up ray) and requires networking synchronization.

4.2.2.3 Camera manager

This manager is responsible for the control of the camera game object. The camera of the game has an RTS perspective meaning that the user can change the viewport by placing the mouse one any of the four edges of the screen. However, by default the camera is *locked on* the player character and moves based on its position on the field. The user can lock or unlock the camera by pressing the "L" key and zoom in or out via the mouse scroll. The camera has two modes, the Earth and the Heaven mode based on where the player character is located.

4.2.2.4 Player input manager

Any player character related input is triggered from this manager. The components of the player character [4.2.1] listen to those events and call the necessary procedures accordingly.

4.2.2.5 Teleportation manager

The teleportation manager is responsible for teleporting the player character between Earth and Heaven. It pauses any movement or input given during the time that is needed in order for the transposition to take place and changes the camera mode accordingly. Furthermore, it silences the player character on arrival at Heaven since players are not allowed to cast skills at that area.

4.2.2.6 Utilities

Procedures that are shared among different modules of the project are also part of the utilities singleton. In addition to that, this manager contains assertion methods that are used to validate and test the game when necessary.

4.2.2.7 Holders

- *Skill book:* Contains the list of skills that exist in game as a map with an id as a key and the actual skill as a value.
- *Resources paths*: Holds the file system paths of the assets that are dynamically instantiated or set in game (skill objects, aoe effects, dragon colored textures .etc).
- *Scene hierarchy paths*: Holds the paths of certain game objects in the scene hierarchy that are used as containers of the actual game objects. This way the scene hierarchy is stays "clean", which is really helpful for debugging purposes.
- *System messages*: Includes the system messages that can be displayed based on the events that can happen during combat (10 kills in a row, suicide, critical hit .etc). System messages can be helpful to players since the visual effects are poor in dragonborn.
- *Colors*: The RGB values of each player slot's color that exists in game is maintained here. Those are used for certain GUI components.

4.3 Combat System

4.3.1 Effect modifiers

Except of the special effects that affect player characters in a specific way, all the other effects are modifying the attributes of the character they were activated on. An effect modifier is a pair of two numbers that modify a certain attribute (depending on the effect type), each in a different way. The first number is the *raw value* which is added or subtracted from an attribute either directly or by the use of a formula. The second one is the *percentage value* which defines the percentage of the maximum value of the attribute that is going to be used the same way. For example lets assume an effect modifier (0.1, 0.2) of a lasting movement debuff effect and a movement speed bonus maximum value of 0.5 which is translated to 50%. This effect will reduces the movement speed bonus to 0.3. 10% was removed because of the raw value and another 10% was subtracted as the 20% of the 50% movement bonus that was there from the start, resulting to a

final 30% movement bonus after the activation of the effect (or -(raw + percentage * attribute's maximum value)). In the previous example the attribute modification happened directly, which always happens with the maximum value of any attribute. However, to modify the current values of the vital attributes formulas are used that are based on the values of certain normal attributes and the effect modifier.

4.3.1.1 Vital modifier formulas

The vital modifiers' formulas that exist in game are the following, where caster is the dragon that used the skill and receiver the character that it was attached to:

- **Damage**: - (raw * (caster's physical power / receiver's physical defence) + percentage * attribute's maximum value)
- **Manaburn**: - (raw * (caster's special power / receiver's special defence) + percentage * attribute's maximum value)
- **Heal**: + (raw * (caster's leadership / receiver's leadership) + percentage * attribute's maximum value)

The percentage part of the modifier is calculated as previously. However, the raw value is based on the physical attributes for damage, special attributes for manaburn and the leadership for healing (health and mana). This way we could create effects that are strict in a manner, like 20% of the maximum health damage, effects that are based on the character's attributes or both. This type of effect modifier is a very powerful feature that can be used for the creation of many combinations of effects.

4.3.2 Skills implementation

To implement the skill system as described at [3.3.2] a skill is separated in different components that manage a different part of the full skill functionality:

- **Model**: Includes the logic of the skill regarding the effect and requirement lists, cooldown, range and mana cost. The procedures for casting, updating the cooldown time and attaching the groups of effects to dragonborns are also part of this component.
- **Target cursor**: Many skills in game are targeted rather than basic. The target cursor is the component that is needed to manage those skills. It maps the mouse position on the screen to coordinates on the terrain. When a targeted skill is being casted, the information about the destination or direction of that skill (depending on the type of the target cursor) is provided by the target cursor to the skill model.
- **Behavior controller**: This component manages the movement of an object over the terrain. It is the part of the skill that defines a skill as a projectile, trap or other types. In addition to that it contains the information regarding which player characters can be affected by that skill (allies, enemies or both) and variables that describe the AoE nature of the skill.
- **AoE controller**: For AoE skills a second object is instantiated at the location where the location they are triggered. This controller describes its AoE behavior and will be attached at runtime when the object is constructed. Variables like the radius, maximum affected character, time to live on the terrain and activation frequency are set based on the information that exist on the behavior controller of the skill. The last two variables are there to simulate AoE skills that exist for a certain amount of time on the terrain and affect the dragonborns that are located within the skills' radius. A specific example is the icy terrain that lives for a few seconds and stuns anyone that walks over it.

4.3.3 Attaching effects

In this section we are going to describe the manner that the effects are attached on players when a skill is triggered. We have already mentioned that effects are components that can be attached on a game object [4.2.1.8] and that a skill has three lists of effects [3.3.2]. When a skill is being cast, the passive effects are attached and activated on the caster dragon's game object. Those effects are usually of buff or healing type but they can be used to simulate recoil from the use of a powerful skill. When the skill is triggered, the ally or enemy effects need to be attached to the characters the skill was triggered on, depending on their alliance state with the caster. The owner of the skill sends an RPC to any client whose character is affected by a skill through the network controller [4.2.1.7] to let them know about the id of the skill and the effect list that they need to attach to themselves. Finally, they attach the corresponding group of effect themselves and use the attributes of the RPC's sender as attributes of the skill caster by accessing the game manager.

5. DISCUSSION

In this section we are going to review this work by discussing the strong and the weak points of the game. We will analyze the modifications in terms of game player or implementation that took place after testing the game, in order to improve the user experience. Finally, ideas or features will be presented, that were never implemented due to either lack of time or poor choices.

5.1 Skill book

The skill book window of the game describes the skills that are maintained on the skill book holder. However, it is poor in a way that the player is not informed analytically about the effects of each skill. There is only the title, cooldown time, mana cost, availability icon, the requirements and a very brief description regarding its effects. The GUI library that was used for the implementation was introduced to the project nearly at the end and because of that reason there was not enough time to get used to it and implement everything as they were supposed to be. Apart from the GUI window there is a more important issue regarding the skill book. Every skill that is part of the skill book was hard coded in the constructor, since at that time it seemed a lot less time consuming even though it was well know it was a bad idea. However, after the implementation and during the testing of the game a lot of time was spent to balance the effect lists of each skill. In the long term the choice that was used in order to save time was holding back the balancing process. In those cases where a lot of data are being maintained and may need to be modified there is a need of separation from the code. That separation could be achieved by JSON or XML files that would keep all those information. In addition to that a parser is needed to transfer those data from the files to the memory when the game starts. Even though the main idea was that constructing the parser in order to use configuration files would be time consuming, in the end more time was wasted on trying to modify the hardcoded data from the skill book's constructor. Finally, another good idea for large scale

projects where game designers that are not experienced with programming may be part of, would be to maintain a database on a machine where all the data would be stored. This way adding, modifying or deleting data would be easy for those people. Then configuration files could either be generated by the data stored in the database when modifications take place or a server could be hosted that could provide responses to the requests from the game modules.

5.2 Camera movement

Another feature that was never implemented the way it was supposed to be it the camera movement. When the camera is not locked on a target that follows it can be moved by placing the mouse cursor on the edges of the screen. This was implemented by calculating the distance in pixels from each edge and when needed movement was applied. However, this requires constantly to check the distances of the cursor from the edges which is a waste of resources. A better way to resolve this feature would be with four invisible GUI windows. When the mouse would collide with those, the corresponding movement would be applied. In addition to that, another way to enrich the game would be to rotate the camera over the Y axis when is located close the corners of the terrain. The way the camera moves and is limited in every edge of the terrain is very straight forward. By rotating the camera towards the center of the terrain would sent an indirect message to the players, informing them that they should probably head to that direction because there is no more terrain left.

5.3 Visual effects

In battle strategy games, users need to be informed about many different events that take place and a large variety of visual effects is a great way to achieve that. Specifically in dragonborn, there are different skills that are used by the players that have different impact on the character that they could be activated on. Some skills modify the health and mana current values that are right in front of the user to see. However, other can affect the maximum values of the attributes that are displayed on the character window. It is extremely difficult for the user to understand what exactly happens they get hit by certain skills and therefore information relevant with the attribute modifications are needed to be provided to the player. It would become really chaotic to display the values that were added or subtracted on certain attributes all the time. Actually, there is no better way for players to learn about the logic behind skills, than memorizing and using the skills by themselves. However, it is very important to help players that are fairly experienced with the game by providing to them information in a smart but efficient way. For example we could have used colors close to red for skills that include damage effects, blue for manaburn effects or a bright color for healing skills. Even though some of the players may not recognize the pattern themselves, it will be easier for them to distinguish the use of each skill. In addition to that, rather specific effects like stun or silence limit the controlling of the dragonborn. However, the player should be notified somehow that such a thing took place. For example, when a dragon is being stunned, its skin color could become icy for the time that the effect lasts. When silenced, a black cloud could be on top of its head. Other features where visual effect would be very useful are the overtime effects and the AoE skills. Regarding overtime effects, players would be aware of the time length and the frequency of those if a particle effect would spawn on every activation. Also if an explosion or similar effect took place when AoE skill are being triggered, players would understand how large the radiuses of those skills are which will help them protect themselves easier in the future. Finally, since numbers are a crucial part of the battle, sometimes number related information could be also useful to be displayed to the users. For example, displaying the health or mana that was added or subtracted after a vital effect would be a desirable feature that would not happen very often, but would make the player happy when it is achieved. There are several ways to provide information of minor or major significance in a game and visual effects are one of the best because except helping the players, they also make the game a lot more exciting.

5.4 Soundtrack

Unfortunately, there is no background music or sound effects in game. A soundtrack could convey the feeling that is the game designer is targeting for the players to feel while playing. In this particular game different soundtrack themes could be provided to the users just like the rest of the game preferences. Proper sound effects could transform and make a lot more interesting and exciting a game the same ways as with the visual effects. Different sound effects on every battle event (skill casting, triggering, player kill .etc) would really improve the user experience by giving leads to the players about what is happening. By adding soundtrack to a game, players are using one more sense while playing without any effort. The more the senses one uses while playing, the better the game experience is.

6. Conclusions

Finally, in this work we designed, implemented and discussed about the dragonborn video game. A multiplayer game that connects users of over the internet and lets them compete with each other. Players control and train a little dragon each, while using skills to fight against their opponents or help their allies. Creating a video game is not an easy task and there are many aspects that need a lot of though. We discussed about the strengths, weaknesses and ways of improving the game. At the start of the project, not too much time was spent on searching for tools that could really speed up the development. This resulted to the waste of a lot of time for refactoring and thus less time spent for testing or adding more features. The implementation part is always what sets the limits when creating a game, since it requires a lot of people to work in a team to achieve a satisfactory result. However, as the popularity of game development grows and the community is getting larger, more tools and resources that do the heavy lifting are being created. Smaller teams are able to create their dream worlds and transmit their feelings to the players. This is what entertainment games are about, the fun and the feelings their players feel. There is no chance that a game that is raised that way, will not grow to be loved by its target audience.

7. References

[1] Blizzard Entertainment: Warcraft 3, The frozen throne. 2003: http://www.wowwiki.com/Warcraft_III:_The_Frozen_Throne

[2] Riot games: League of Legends. 2009: http://www.leagueoflegends.com

[3] Jesse Schell. 2008. The art of game design: a book of lenses. http://artofgamedesign.com.

[4] Unity engine: http://unity3d.com, Sept 2013

[5] Photon Unity Networking: http://www.exitgames.com/en/PUN, Sept 2013

[6] Daikon forge GUI: http://www.daikonforge.com/dfgui, Dec 2013

[7] Unity asset store: https://www.assetstore.unity3d.com. Sept 2013

[8] Unify community wiki: http://wiki.unity3d.com, Sept 2013

[9] C sharp design patterns: http://csharpdesignpatterns.codeplex.com, Oct 2013

[10] Burgzer arcade, Hack and slash RPG: http://www.burgzergarcade.com/hack-slash-rpg-unity3d-game-engine-tutorial. Oct 2013

[11] Stormtek, creating an RTS in unity: http://stormtek.wordpress.com/2013/04/16/creating-an-rts-in-unity, Oct 2013