Ascendance : A Platformer for Computers

By: Philip Scott Beauchamp and Joseph Grant Plaster

California Polytechnic State University

Advisor: Dr. Zoe Wood

June 9, 2015

Abstract

Ascendance is a 2.5D platformer adventure game for computers, developed in the Unity development environment over two quarters to apply computer graphics and software engineering principles to the programming and iterative development of a game. Players encounter enemies which they must defeat or avoid while they explore the level for the required objects that lead them to victory. Testing of Ascendance required several iterations of play testing and surveys by various testers with and without gaming backgrounds, and was followed by rapid refinement of game mechanics and aesthetics. The result is a fully functional demo that features a sophisticated start screen, backstory, tutorial, and single level, with the ability to easily expand the game to encapsulate more levels and new features.

All work listed in this project was in joint collaboration between Philip Scott Beauchamp and Joseph Grant Plaster, with the exception of "Section 4.10 - Particle Effects."

1 Introduction

Each game is unique in its own way, but one thing they all have in common is their connection to the Computer Science field. While game developers seek to be at the cutting edge of the newest technologies to create a better, and perhaps more realistic game, the most successful games today combine elements of meaningful play, unique gameplay, and an intriguing story [7]. Most problems that determine whether a game is successful or not stem from heavily focusing on one of these elements and ignoring others, thus resulting in an imbalance [7]. Ascendance is an attempt to study the full process of game development and create a game that successfully captures and balances all of those elements.

Ascendance is a 2.5D platformer adventure game that tries to give players a meaningful gameplay experience, implement unique gameplay mechanics, and maintain an immersive and intriguing story. Ascendance was created in the Unity[11] (Personal Edition) development environment by Joseph Grant Plaster and Philip Scotty Beauchamp over two three-month quarters, with very brief exposure to Unity before production began. The story of the game follows a boy named Kain, who was chosen by the Keepers of Light, the gods of his world, Zorah. The Keepers of Light protect Zorah's source of light and good energy – a crystal called the Vitrelux. The Vitrelux was corrupted and shattered by a monster of darkness called Atrox, who began to summon an army of shadow creatures to control Zorah. The Keepers of Light use the last remaining light energy to bless Kain because he is pure-of-heart, so that he may save his home. The player's goal is to explore the dying planet, find shards of the Vitrelux crystal, and cleanse them while defeating any enemies that get in their way. Upon successfully cleansing all crystals, Zorah is saved and the player is congratulated and returned to the start screen.

During the production process, Grant and Scotty developed a “production pipeline” for importing digital art assets, interaction scripts for unique gameplay mechanics built on top of the commercial Unity game engine, and a functional tutorial and first level of the game.

2 Related Work

There are several commercial video games that influenced the gameplay and art style of Ascendance. The game’s mechanics and character features were influenced by games such as Risk of Rain and the Super Mario Bros. series. The game’s art style was also influenced by Castlevania: Symphony of the Night, and several other modern games.

2.1 Super Smash Bros.

Super Smash Bros. is a popular action-platformer game developed and released in 1999 by Nintendo, which spawned a series of sequels that continue to be released today [8]. The Super Smash Bros. series was essential to popularizing modern 2.5D platform games. The initial style and gameplay mechanics of Ascendance follow a similar pattern of 3D models constrained to a 2D playing field, where the character can pass through a platform from below, but is stopped when colliding with a platform from above.

Figure 1: Super Smash Bros. Brawl Gameplay[9]

2.2 Risk of Rain

Risk of Rain is an action platformer with roguelike elements [3]. In Risk of Rain players only have one life, and must fight their way as far as possible before having to start over. Risk of Rain's dark overtones and overall difficult gameplay had great influence over the development of Ascendence. The value associated with having only one life is captured in Ascendence just as it is in Risk of Rain because dieing means starting over, from the beginning.

Figure 2: Risk of Rain[10]

2.3 Castlevania

Castlevania is an action-platformer game released in 1986 by Konami [2]. Castlevania and its sequels follow vampire hunters as they hunt for Dracula. The dark theme for Castlevania and its many sequels was a major influence in the inception of the Ascendance game story. Like Castlevania, the story for Ascendance features a young protagonist in search of dispelling a dark shadow from the world. The fire and shadow attacks from the main character and enemies in Ascendance also draw from Castlevania, as they try to capture a sense of magic and mysticism - both very key to the intrigue of playing a Castlevania series.

Figure 3: Castlevania - Symphony of the Night[6]

2.4 Other influences

Other games that influenced Ascendance in smaller ways include:

- Defense of the Ancients: Free online character were used for the enemies in Ascendance [5].
- Destiny: The art style influenced the look of Kain, the main character of Ascendance [1].

3 Why Unity?

Ascendance uses Unity as its game engine. Unity is a multi-platform rendering engine for creating interactive 2D and 3D content [11]. Unity features a 3D scene view similar to that of 3D modeling programs, a hierarchy to organize game objects, a built in physics system that can be customized, customizable particle effects support, and is compatible with scripts written in C#

or Javascipt. All of these provided supports lifted a burden of creating physics or particle engines within a game, leaving development to focus on the desired mechanics and visuals of the game.

Ascendance utilizes the scripting support to apply scripts as behaviors to game objects, defining how they interact with the respective engines and with other objects in the world. Unity provides a standard library for vector and quaternion math, coroutines to make multiple rendering calculations per frame, and physics operations to alter or override the existing physics engine that Unity provides[12].

Ascendance takes advantage of the powerful tools the Unity provides, and alters them in its own unique way to provide an immersive gaming experience with unique mechanics. Learning to use Unity took some time, but ultimately saved time in overall development, and was a valuable experience to developing games in a completely new environment.

4 Implementation Details

4.1 Game Story

Before any production on Ascendance began, one aspect must be laid out and thought through: the story. Ascendance's story began as scattered ideas written down on paper, until something stuck. Once a very basic idea was struck, more details followed until eventually the basic outline of a setting and plot were created. This outlining of the story was the beginning stages of the creation of Ascendance's story.

Creating the storyboard of Ascendance's story was the next step. This involved creating a comic-style, illustration of the story. As shown in the figure below, each frame of the storyboard painted a picture of a part of Ascendance's backstory. Drawing out frames of the

backstory helped take the ideas scratched out on paper take form and have life. With a storyboard the story begins to take shape, and a clear vision can be seen of where the story is going. The amount of help that a storyboard provides seems trivial and a waste of time, but it is quite the contrary. Laying out Ascendance's storyboard set up the beginnings for our production, and made choices for the setting and look and feel of our game much simpler right from the start.

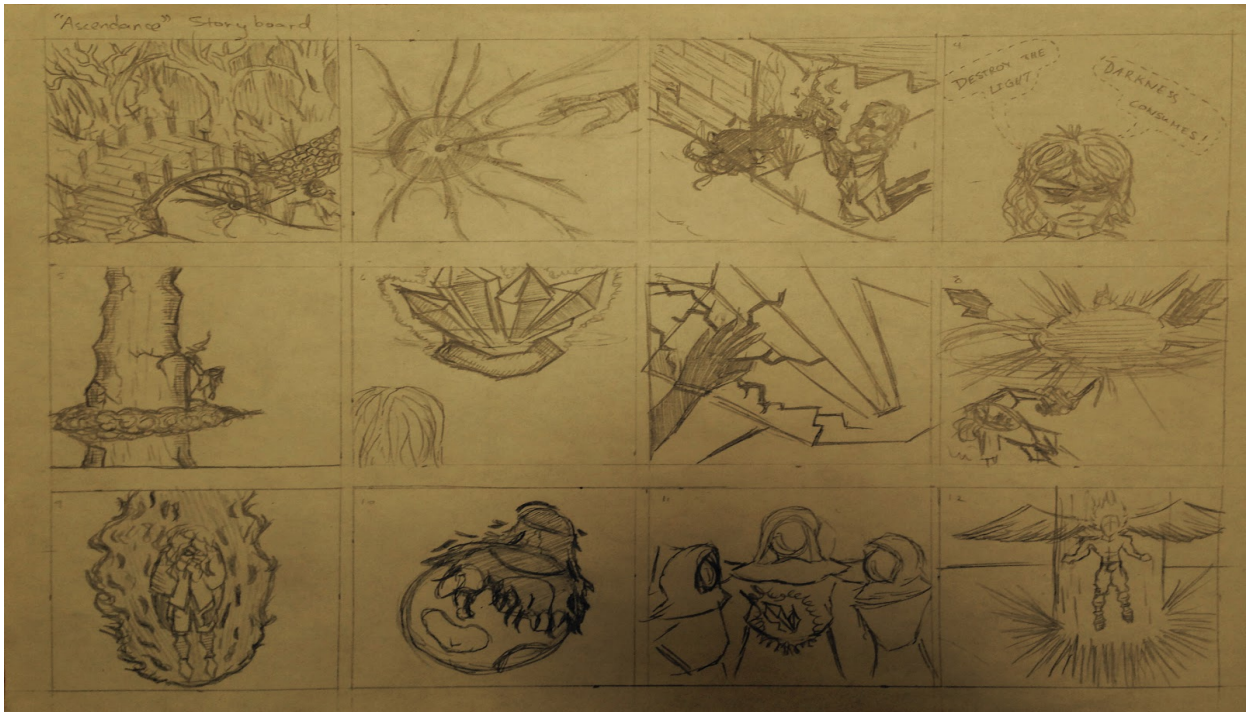

Figure 4: Ascendance backstory storyboard

Developing a story engages players to not only play the game, but be involved with what they are doing on a more emotional level [4]. The backstory reveals how the game world of Zorah fell into the turmoil that the player is placed in, and gives context for what they are doing. The story tries to engage players to assume the role of Kain, and feel accomplished when the cleanse the crystals and save the world. Not only context, but a sense of purpose is given to the player to make their playing experience meaningful.

Creating Ascendance's story was the first step in the implementation process, and arguably the most important. It lays out a clear vision for the game, and ensures the players have a purpose in playing it.

4.2 Start Screen and Game Levels

The first look players have at Ascendance, and what really sets the tone of the game story and play, is the start screen. A gentle fade from a smoky background with light firefly-like particles in the air into a title and game buttons is complemented by an eerie soundtrack. The start screen utilizes a Canvas element that Unity provides; the Canvas provides a screen-space overlay of UI elements over the screen. Ascendance has a text element for the title and two buttons, one for the tutorial level and another for the main game level. The buttons are handled using C# scripts, and upon being clicked will trigger to assigned functions that will start their respective levels.

Figure 5: Start Screen

Upon clicking the button for the tutorial, the Canvas elements gently fade away, and a wall of text crawls from the bottom of the screen. The text is scripted on a timer when the alpha value of the color on the title buttons reaches 0, and the position is updated along the text's y-axis. Unity utilizes a built-in function called Update, which is called every frame of the game, and the text is iterated there by a provided variable Time.deltaTime - the difference in time between each Update call, so that the text will move a little bit every frame until it reaches a certain position. Once that position is reached the tutorial is loaded. The tutorial level has a basic managing script that changes the text elements on a Canvas on a timer to display introductory instructions to the player. The player is showed the several basic controls, using "A" and "D" keys to move left and right, the spacebar to jump, and clicking the mouse to shoot a fireball. After the basics, the player is brought to a crystal and encounters an enemy, accompanied by instructions to cleanse the crystal and defeat the enemy. Once the crystal is cleansed the player is notified that they are ready and transported to the main game level.

Figure 6: Tutorial level

The main level also contains a script that manages the game. The manager script is in charge of calculating which crystals, among the five scattered through the level, are cleansed. The manager also maintains the spawn rates of the enemies. The spawn points of the enemies are contained in arrays that are passed to the scripts from the game engine. The manager iterates through the arrays to spawn flying enemies, which are only spawned once at the beginning of the game, and continues to check the spawn points of the grounded enemies. Once the player is within a varying distance each spawn point, the enemies will begin spawning and will continue to spawn every seven seconds or so while the player is nearby.

Figure 7: Main level

Once within range of a crystal, the lights in the game and around the crystal begin to have a strobe effect, and the spawn rate of nearby enemies will increase to every five seconds while the player is cleansing the crystal. The light strobe, as well as the color of the crystals' emitted particles changing as the crystal nears its fully cleansed state, was a design choice to increase the intensity of the act and show the player visual feedback and progress.

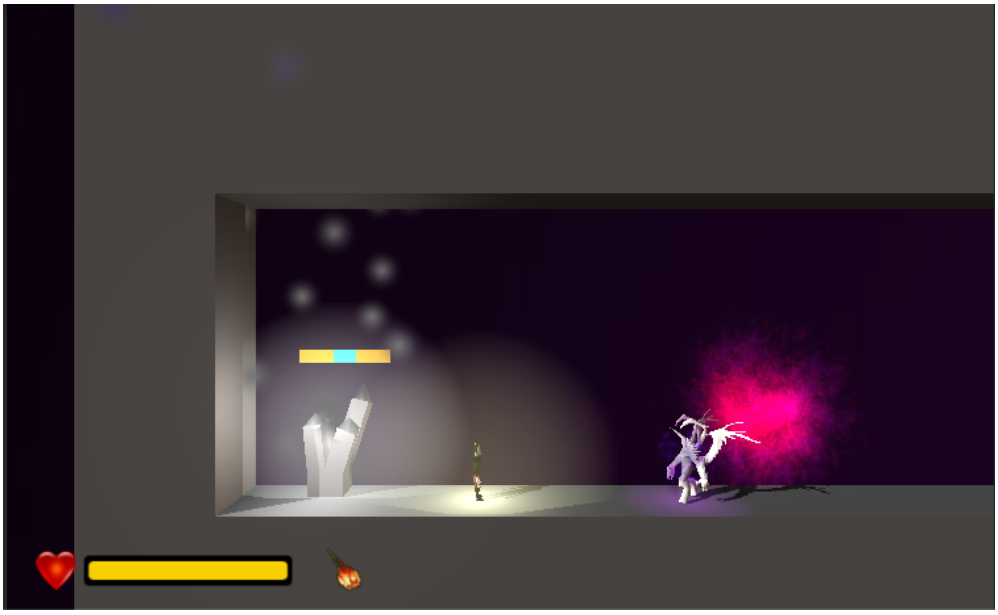

Figure 8: Crystal strobe effect

The player also receives a new ability in the level - the ability to use a grappling hook for navigation. The grappling hook is toggled with the “E” key, and is vital to navigating the level (see section 4.4 Scripting for more details on the grappling hook).

After cleansing all five crystals in the level, the player is congratulated on their victory and returned to the game’s start screen. These levels were the foundation of the game and were very important to learning about level design, setting the tone for a game, introducing a story and introductory gameplay to a new player, and managing a multitude of interactions between various game objects within the main game loop.

4.3 HUD

Ascendance’s HUD does not take up much of the players screen space; it is is simplistic yet gives the player the necessary information needed to get through the game. The HUD was created using a Canvas element in Unity. The Canvas in Unity is an area in which UI elements (text, sprite, buttons, etc) can be placed in. This Canvas element corresponds to the screen

space, therefore wherever these elements are placed in the canvas is where they will show up on the screen[12].

Once a canvas was created, and renamed HUD Canvas, elements were placed into the canvas one by one. Ascendance's HUD Canvas is comprised of a small amount of UI elements in order to make the screen less cluttered and more simplistic. This allows for the user to focus more on the gameplay and visuals, rather than be distracted by an over-the-top HUD. Ascendance's UI elements in the HUD Canvas include a heart shaped sprite, a yellow slider bar used as a health bar, two sprites for the fireball and grappling hook, and text placed in the bottom right to alert the player of the amount of crystals they have cleansed.

With all of these elements in place instances of them were placed in various scripts to modify the logic associated with them. In the player health script as the player's health decreases the range of a light source parented to the player also decreases, and this is reflected by adjusting the slider bar based on the player's current health. The fireball and grappling hook sprites are simple made transparent or opaque based on whether or not the grappling hook or the fireball is selected.

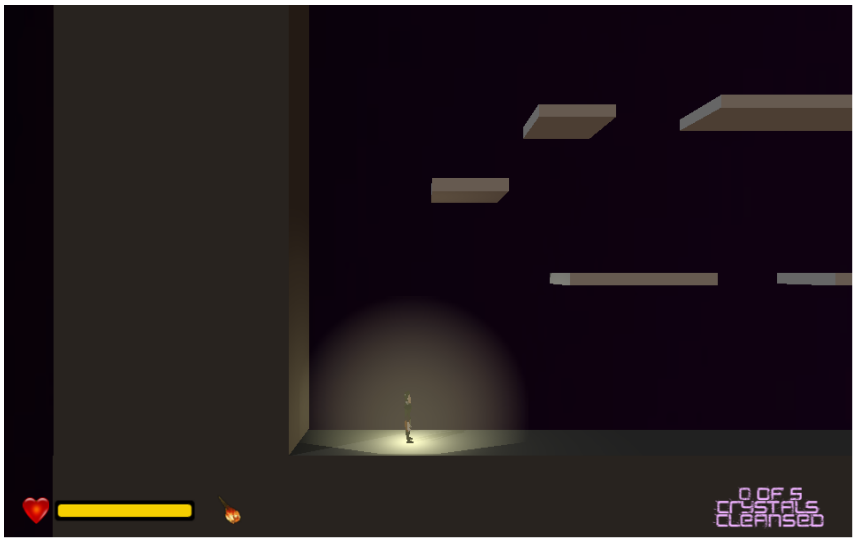

Figure 9: HUD with fireball selected

Pressing the ‘E’ key switches between the two projectiles and either makes the fireball or grappling hook sprite transparent depending on which one the player is switching to. The text in the bottom right hand corner is initialized to display “0 Crystals Cleansed”. In the game manager script there is a variable which represents the amount of crystals left. This variable is used to determine if the player has reached an end state (if crystals left equals zero) and is also used to update the tet ui in the HUD Canvas.

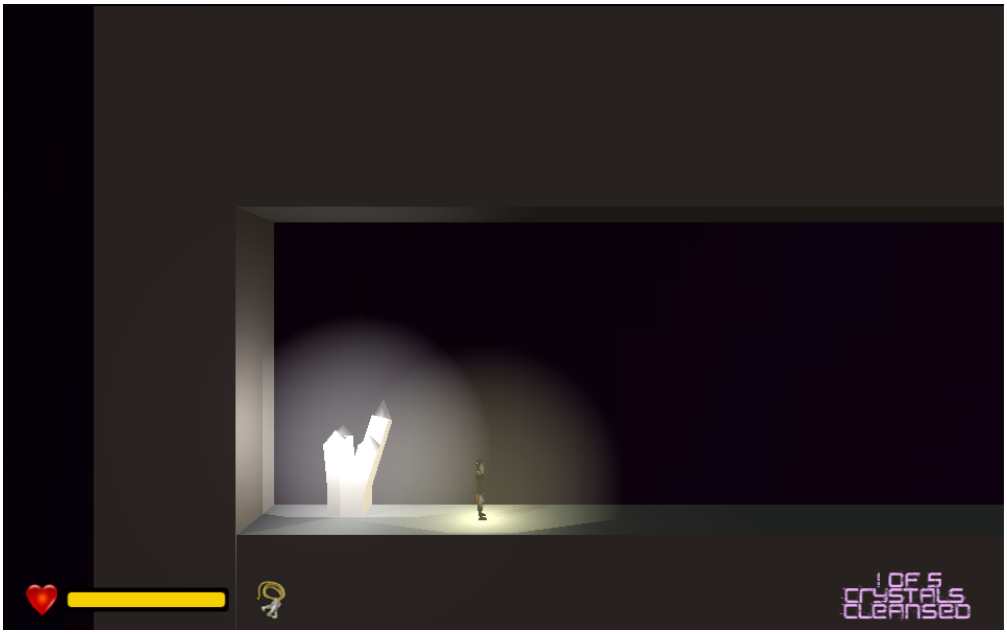

Figure 10: HUD with grappling hook selected, 1 cleansed crystal

4.4 Scripting (Movement - Vectors and Quaternions, Grappling Hook)

While Unity’s engine is the backbone of Ascendance, C# scripts are the brains and govern nearly all interactions and behaviors in the game. Scripts operate with some common standard library functions such as Awake or Start for initializations, Update for calculations made every frame, and Fixed Update for calculations made for every physics step[12]. Player movement is determined by retrieving the horizontal axis that the player should move in; this is determined by a key press and the result is a -1, 0, or 1 for moving left, no movement, or right,

respectively, and is in the x-axis. The position is updated while the player holds down a key to move the transform position along the axis for movement.

The fireball attack calculates a vector between the mouse - captured from screen space and translated into world space - and the player when the mouse is clicked. This vector is used as the direction vector for the fireball to move towards when it is instantiated at the player's position, allowing the player to shoot a fireball in any direction of the mouse click regardless of which way the player is facing. When the grappling hook is applied, the same algorithm is applied to calculate the direction in which to shoot, but the hook is fired with a line renderer as the rope to connect the hook and the player. Once the hook collides with a wall, lerping - linear or spherical interpolation between two points - is applied to gradually move the player transform's position to that of the hook. Lerping occurs over several update frames with the starting position of the player being 0, the position of the hook being 1, and the interpolated values in between are increased by Time.deltaTime intervals; the difference in time between frames are fractions of seconds and allow for smooth interpolation between two points to create a feel that the hook really dragged the player to its position [12].

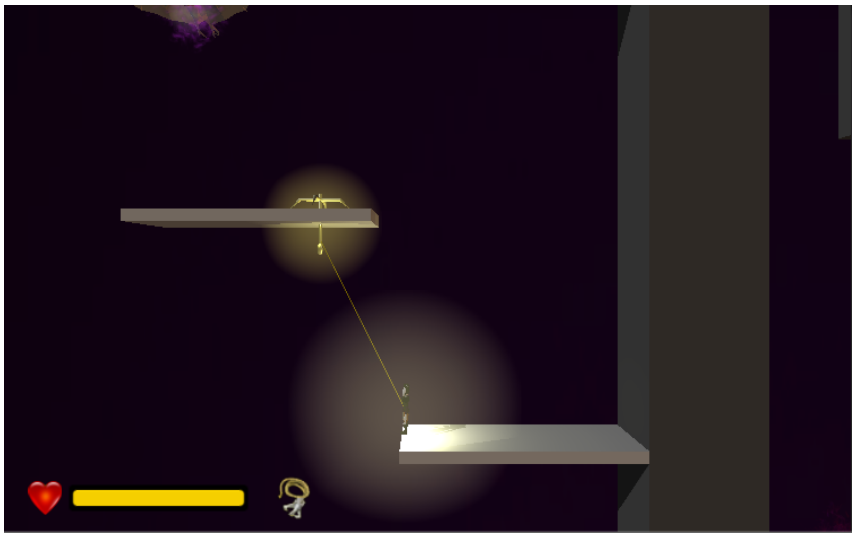

Figure 11: Grappling Hook before Lerp is applied

The color changing strobe effects on the crystals and the lights, as well as the movement for idle flying enemies operates on a sinusoidal curve that is scripted to be called every frame, to simulate a strobing or oscillating effect. Flying enemies are also lerped, spherically, around their game object's pivot point as the player enters their range. The difference is, their Quaternion rotation is lerped - as opposed to their Vector position - to position them to face the player.

Scripting was a very powerful and useful way to govern behavior for game objects. It provided a chance to build upon programming skills already learned, while learning how the scripts interacted with each other and with the other components of the game such as physics, animation, or artificial intelligence to create a sophisticated game with enhanced graphics to display and unique gameplay mechanics to provide for a fun experience for players.

4.5 Physics (Collision, Forces)

Unity provides a physics engine, which made certain applications to the game objects, such as gravity and object collision, very simple. Colliders can be attached to every game object, and can be distinguished as a "trigger" or regular collider. Triggers do not perform physics operations, they simply capture that a collider on an object intersected with something [12]. In that case, the provided standard library function - OnTriggerEnter or OnTriggerExit - was used to instruct specific behavior. Triggers were key for detecting if the player intersected with the ground or a platform to allow them to jump again, see if the player or enemies had been hit by an attack and should take damage, and to see if the grappling hook had collided with something and should move the player to its position.

Non-trigger colliders prevented the player and enemy game objects from falling through the ground or platforms whose gravitational force was 0 so it was unaffected by gravity. Non-trigger colliders also prevented the player from clipping through walls when using the grappling hook.

“Forces” are other operations that are provided in Unity’s standard library; forces allow user-defined physical forces to applied to game objects for movement within the context of the physics already defined in the game [12]. Forces were applied to the player game object when the spacebar was pressed, to launch the Kain into the air and simulate a jump. The force dissipates when it reaches a peak as gravity is constantly being applied as it would in the real world. Forces were also applied to instantiated instances of the grappling hook to simulate the hook being thrown out across the map.

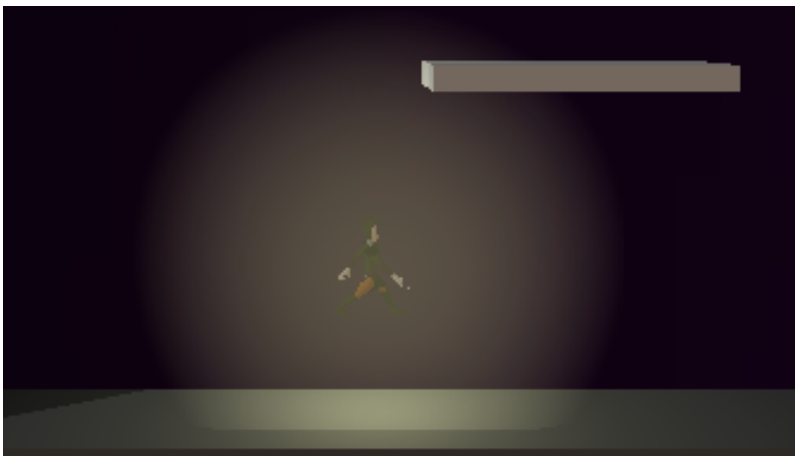

Figure 12: Forces being applied to create jump

4.6 Prefabs

With the simplicity of Unity’s drag-and-drop style of placing game objects into a scene, Unity’s provided ability to create Prefabs made duplication of game objects easy and efficient. Prefabs allowed game objects to be created with specific parameters that apply to each

instance, where that instance can be saved and duplicated [12]. For instance, each enemy type is an instance of a single game object that contains behavioral scripts; this allows for the duplication of enemies that all behave in the same way. Similarly, each instance can share the same scripts for behaviors or physics calculations, but hold its own properties for shape and size. All of the level platforms were created from a single platform prefab; they all behave the same way with the player, enemies, and grappling hook, but they each are a different size and dragged into their own position in the world to create the levels.

Prefabs were useful too when wanting to make changes to all instances of an object. If a script was added, simply clicking on an “Apply” button would apply the script to all instances of the prefab and save it so future instances would also share it. However, the fault of prefabs lies in changing them. When a change was applied, an undo command would not work, so on more than one occasion a change was made by accident that altered the way an object spawned in the world and the object behavior had to be redone completely from scratch.

4.7 Animation State Machine

Unity provides a very unique and amazing tool known as the Animation State Machine to their users. This tool was used to help define and organize the different animations for every animated Game Object in Ascendance.

Ascendance created animations in the Maya 3D Rendering Program for both the main character, Kain, and the main demon mob enemies. The animation state machine is a complete graphical representation of the animations for a game object [12]. For example, one game object may have an animation “idle” state, a walking state, a running state, and a jumping state. These states all have relationships with one another and can be linked together using the animation state machine. These links provide logic as which state can reach one another. In Ascendance,

the main player game object had two states: waking and idle. The two states were linked together and a parameter was placed on the state machine called isWalking. This parameter is a boolean value that is used to set a condition on the animation state machine. The player game object can only transition from the idle state to the walking state when the isWalking parameter is set to true. This parameter is set to true in the player movement script, when the user makes the player move in the game world.

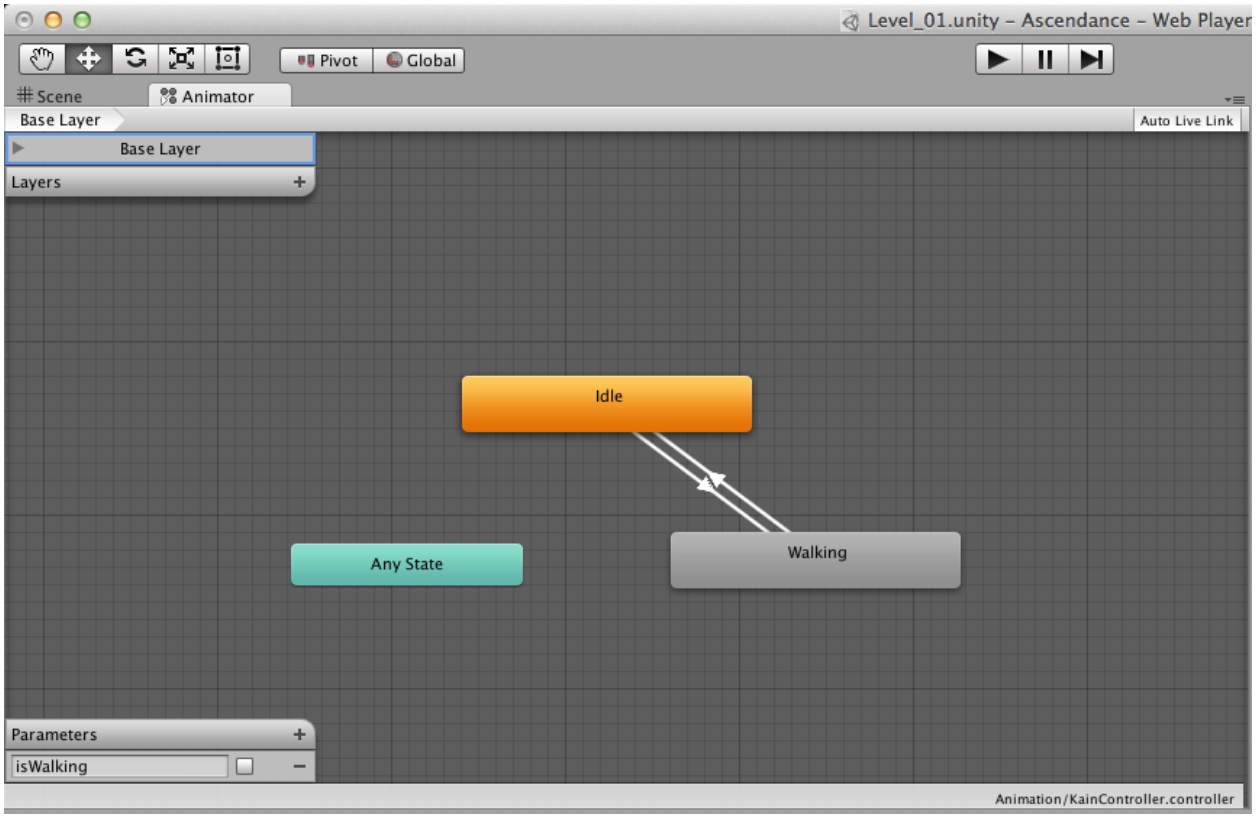

Figure 13: Animation State machine controller

4.8 Artificial Intelligence

Sophisticated AI was not a major focus for Ascendance, but simple navigation and interaction was built using scripts and Unity's built in support of navmeshes. Navigable surfaces were defined during development and labeled as static in the Unity editor. A navmesh agent

was attached to all grounded enemies, and a plane of navigation was calculated and baked into the agents to distinguish to the game object where was navigable by the agent [12].

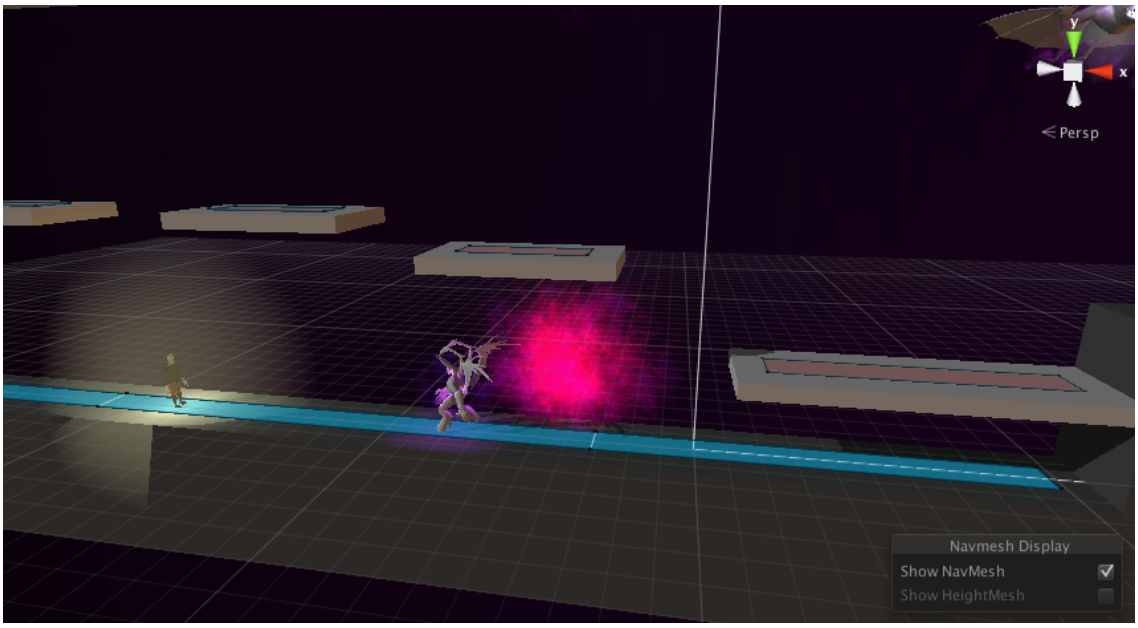

Figure 14: Navmesh displays navigable planes for enemies on stage

Simple scripts calculating the distance between the player and an enemy would set the navmesh destination to the player's location when they were in range of the enemy. With animation and navmeshes, the enemies followed the player to attack them.

For flying enemies, an idle state left the game object oscillating on a sinusoidal curve in the x-axis. Once the player was within range, the game object was spherically rotated to face the player and their shadow ball attacks were instantiated on a timer to fire at the player. Even though the AI for Ascendance was simple, it made for a somewhat difficult game and provided good insight for how much even a little bit of AI does for a game. Scripts and navmeshes, combined with ranged enemy attacks made for a good challenge and a variety of enemies to keep gameplay fresh for the player throughout the game.

4.9 Gameplay testing

Gameplay testing was a major focus in the development of Ascendance. Gameplay testing began in the later stages of development of the game.

Unity was used to create a build of Ascendance, which could be tested by various groups of people. Then a survey was created for every person to take after or while they played Ascendance. This survey gauged their familiarity with video games and also asked what specific aspects of the game they liked or disliked and how they would improve it. In total around twelve to fifteen people were questioned on the first build of Ascendance. Results to the survey of the first run of play testing were then recorded in a Google spreadsheet. In general, people found multiple bugs associated with the grappling hook, the flying enemies, and collisions with platforms and walls.

With this information, improvements were made to Ascendance so that the grappling hook behaved more appropriately (slowed down and more fluid), platform and wall collisions were less buggy, and so that all other play tester's complaints were attended to. Then, another build of Ascendance was made for a second round of testing. With this build play testers were asked similar questions to the first survey, and at the end were asked if the games changes from the previous build were improvements or not.

This play testing and surveying proved to be one of the most important aspects of the overall development of Ascendance. It allowed for an unbiased judgment and analysis of our game. Having multiple minds analyze the game meant more bugs being discovered and many more points of view being seen on issues with the game. Shedding a different light on these problems allowed for the game to grow and improve at a very efficient rate.

4.10 Particle Effects

All particle effects in Ascendance were created using Unity's built in support for particle systems. Unity natively supports the creation of particle systems as game objects. Once a particle system is created, parameters can be modified to customize the size, shape, and lifespan of particles. Particle effects were used to create Kain's fireball attack, enemy shadow ball attacks, portals to distinguish enemy spawn points, and particles emitted from crystals.

For both the fireball and shadow ball attacks two particle systems were used, one for the core of the balls parented to another to create a stream to follow behind the ball. The core system created particles from a spherical emitter, with a smoke textured applied to each particle for the cloudy atmospheric look of fire. The stream has a longer lifespan for its particles and emits from a cone shape. Both textures are run through an additive particle shader to calculate the light that is reflected on each particle and adjust the color and opacity over each particle's lifetime; the the spawned rotation of each particle is random between two user-defined curves to create a naturally looking generation of particle rotations. Scripts were applied to the parent systems to move the particle systems, adjust color and speed of particles, and determine collisions with other game objects.

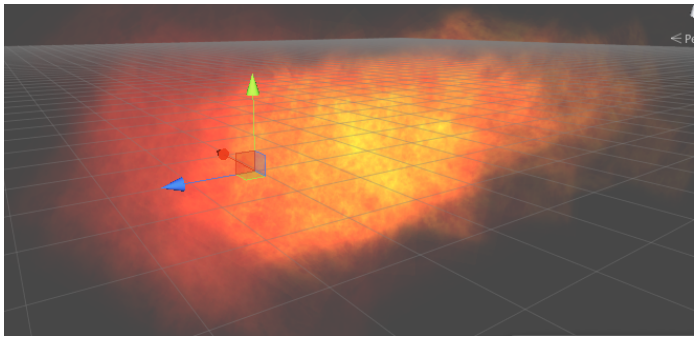

Figure 15: Fireball effect

The portals to distinguish enemy spawn points were created with particles emitted from a hemisphere shape. Particles were textured with a tessellated spark textured, shown in the figure

below, and customized to emit a red and purple color so as to represent another dimension that the enemies are coming from. Particle sizes were large with short lifespans, and again the rotations were randomized upon the start of the particle lifespan. Portals were placed throughout the game to notify players where enemies might spawn from, but no scripts or other defining behaviors were attached.

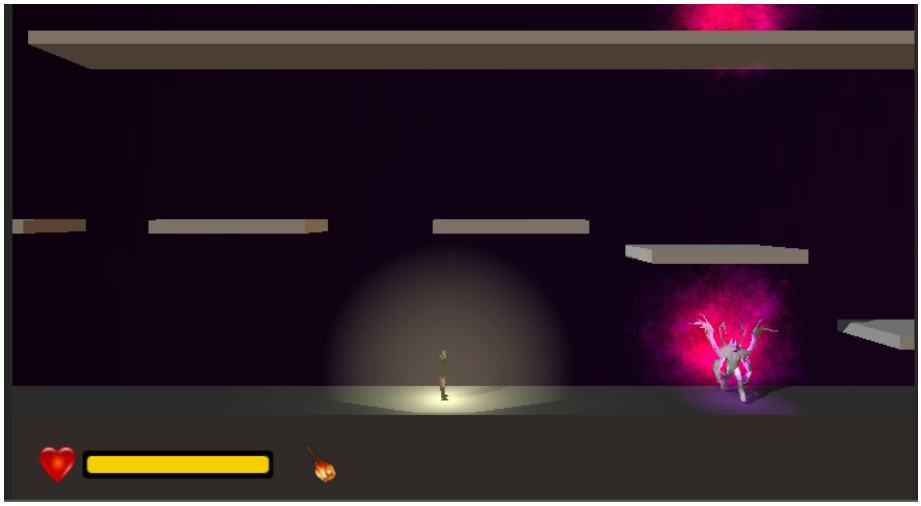

Figure 16: In-game portal effect

Lastly, the particles on the crystals were standard particles from Unity with no textures. The standard particle gives the appropriate look to the sparkles needed from the crystals. The systems to create them were attached on each crystal and angle to emit upward in the shape of a cone from the base of the crystals. Scripts attached to them calculate the player's distance to determine if the crystal is being cleansed. As the crystal is being cleansed, the color of the particles is linearly interpolated from the dark purple color to a pure white, and particle emission stops upon completion of cleansing.

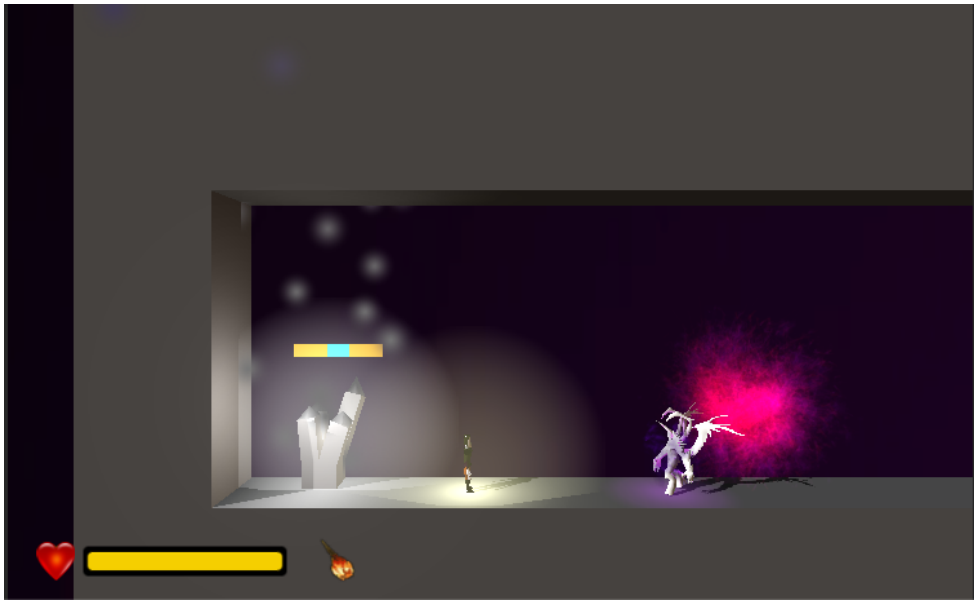

Figure 17: In-game crystal particles changing color

Learning how to create, use, and govern interaction between particle effects was very useful to creating very appealing aesthetics for Ascendance. Not only are the mechanics unique and the story engaging, but with the addition of particle effects the game is much more vibrant and alive.

5 Results

At the end of two quarters, Ascendance has one full playable level, a main menu, and a tutorial level with a short rolling description of the game's story. The game has been built and released via Web Player Streaming, and has been tested using any web browser except Google Chrome.

A production pipeline for creating levels in the game is set up and only requires for there to be very simple dragging and dropping of walls, platforms, crystals, and enemy spawn points throughout the level.

5.1 Conclusion

Unity provided many time saving techniques to help speed along the production of Ascendance. It's 3D model importer and simple drag and drop functionality allowed for a more visual based programming of the world of Ascendance. Scripting on top of game objects usually went fluidly, and made production much quicker.

On the other hand, Unity have issues that at times slowed down the development of Ascendance. One main issue was with the text editor provided by Unity. The text editor was sub par in comparison to others such as intelliJ or Sublime. Another issue was with the lack of any version control within the program itself. Setting up Unity to work with git proved to be a pain, as it did not integrate very well and data was often lost; a built in version control would have been very convenient. The use of prefabs was nice to create many duplicate instances of single objects.

The overall experience with Unity was mostly positive; Unity allowed for a quick development of Ascendance while also providing many resources. Unity is recommended for any future development of games in anyone's future.

6 Future Work

When Ascendance was first envisioned there were many aspirations for the game. Given the time constraints of only 6 months, many of these hopes and aspirations could not be met.

In it's current state Ascendance is not nearly the game that it could be; there is only the start screen, the tutorial level, and a single main level. The basic layout and building blocks for creating more levels is there, and could easily be used to expand Ascendance into a more complete game.

Also, more variation to the game's enemy types is desired as there are only two types. Some potential for desired additions to the enemy types include: a smaller, quicker enemy type that takes less hits to kill and does small damage to the player, and an enemy that can shoot projectiles that track down the player. In the future, a boss is also desired. When the player reaches the final crystal and cleanses it a mini-boss should appear at each stage of the game and Atrox will appear as the final boss at the very end of the game.

As of now the only two attack modifiers are the use of the fireball attack or the grappling hook. In the future, a diverse system of unique power ups will be added into the game. This will range from offensive modifiers - including lightning attack, heavenly blasts, and rays of light - to defensive consumables such as a temporary shield, increased health regeneration, and faster mobility.

Another mechanic to be added to Ascendance is scaling the difficulty of the game. This difficulty will not be on a level to level basis, rather it will be on a basis of time. The more time that passes since the start of the game, the harder the game becomes. Enemies' health will scale based the time elapsed, and the rate at which these enemies spawn will increase. This will be balanced by a slight increase to the amount of damage the player does and by an increase in the amount of power ups that are dropped.

Lastly, a scoring system is desired in future installments of Ascendance. While not allowing players to save their progress through each level, the game can calculate scores based on the speed of level completion, power-ups acquired, enemies defeated, and extra points that can be found throughout hidden or hard-to-reach areas in the game. The scoring would show players how well they are playing the game and allow them to compare their score against others, raising the level of meaningful play they are receiving as they will then be in competition against others.

References

1. Bungie Inc. (2014). Retrieved from http://www.destinythegame.com/.
2. CBS Interactive Inc. (2015). Castlevania Franchise. Retrieved from http://www.giantbomb.com/castlevania/3025-86/.
3. Chucklefish Ltd. (2012). Risk of Rain. Retrieved from http://riskofraingame.com/.
4. GoAnimate. (2012). What is a Storyboard and Why do you need one? Retrieved from http://resources.goanimate.com/marketing/what-is-a-storyboard-and-why-do-you-need-one.
5. Playdota. (2010). Retrieved from http://www.playdota.com/about.
6. Romsdrive. (1997). Castlevania: Symphony of the Night review. Retrieved from http://romsdrive.com/retro-review-castlevania-symphony-of-the-night/.
7. Salen and Zimmerman. (2003). Rules of Play. Retrieved from https://www.brainpop.com/educators/community/wp-content/uploads/2013/10/MeaningfulPlayZimmerman.pdf.
8. Smashpedia. (2015). Super Smash Bros. (game). Retrieved from http://supersmashbros.wikia.com/wiki/Super_Smash_Bros._%28game%29.
9. SSBB Gameplay. (2007). Retrieved from http://supersmashbros.wikia.com/wiki/File:SSBB_Gameplay.jpg.
10. Steam. (2014). Risk of Rain on Steam. Retrieved from http://store.steampowered.com.
11. Unity (Version 4.6.1) [Software]. (2014). Unity Technologies, Novell Inc., NVIDIA Corporation. Retrieved from http://unity3d.com/get-unity.
12. Unity Technologies. (2015). Unity Manual. Retrieved from http://docs.unity3d.com/Manual/index.html.