Technical University of Denmark

02823: Computer Game Prototyping

Knight Wars: Attack of the Bones

Christian Bruun Nielsen - Student no.: s153867
Jeff Gyldenbrand - Student no.: s202790

December 14, 2020

Contents

Introduction **2**

Game Description **3**
- Game Rules . . . 3
- Desired Features . . . 4
- Technical Challenges . . . 4

Implementation **4**
- Technical design . . . 4
- Scripts . . . 5
- Design reasons . . . 7
- Implementation issues . . . 7

Analysis **8**
- Iterations and Feedback . . . 8

Conclusion **9**

References **9**
- Game Manual . . . 9
- Technicalities, coding . . . 10
- Sprites, images, art . . . 10
- Games . . . 10

Appendix **10**

1 Introduction

The first generations of gaming featured many iconic side-scrolling games, which defined the genre of adventure games. Games like Super Mario Bros, mastering the adventure platforming with power-ups and enemies, and Metal Slug, being evolving the genre by making it a more action adventure game with plenty of enemies to fight. Mixing the action adventure game genre with a fighting game resulted in games like Streets of Rage and Dungeons & Dragons: Shadow over Mystara. These games created smaller arenas which would be filled with a defined amount of enemies of several varieties, where players would defeat all these enemies before moving onto the next arena. These side-scrolling action fighting games lost the emphasize on the adventure tropes like platforming and finding keys and secrets (but these tropes still shows up for short sections in order to mix up the flow of the game), and became know as the genre Hack-and-Slash (with the name based on the amount of fighting the player have to do in order to defeat hordes of enemies). Many of these games also adopted a 2.5D playing field, which kept the side-scrolling view but added the ability to move on the Z-axis as well. This was done since these games shifted the focus from platforming to fighting, which required larger and flatter combat arenas.

During the late 90's and early 2000's the wider adaptation of internet happened, and websites sporting many flash-games started showing up. These games tended to stir towards original takes on already defined genres, innovating and refining tropes and ideas. One development group, which would later form The Behemoth video-game development studio, had their start on one of the website Newgrounds where many of their flash-games were side-scrolling Hack-and-Slash games. They would move on to form their studio in 2003, and in 2008 they released Castle Crashers.

Castle Crashers is the main inspiration for the game developed in this project, with its hand-drawn textures and cartoon-ish sprites, dark humor and fine tuned fighting gameplay. It features a long re-playable campaign with many different biomes, enemies and bosses to fight. Starting the game, the player will have access to a small selection of characters with unique appearances but all follow the same move set, with the selection allowing new players to team up with friend for up to four player cooperation, with each player having a unique character. All of these characters have their own attributes and experience amount, which is used to level them up and boosting these attributes for harder difficulties and levels. Playing the campaign also unlocks new characters and combat items (items like bows or shovels used to attack or manipulating the environment), allowing for better differentiation between players.

In this assignment a side-scrolling Hack-and-Slash game *Knight Wars: Attack of the Bones* will be created, taking great inspiration from Castle Crashers and the hack-slash genre. The process of creating this game will be divided into prototype milestones, and each of these will described in detail regarding intend and implementation. An analysis regarding the playability of the game in then performed, followed by a short discussion of feedback from play tests before concluding the project.

2 Game Description

Creating a prototype of *Knight Wars: Attack of the Bones* takes several steps, the first of which is to come up with the design of the game. This requires specifying the objectives and game rules of the game, providing an overview for the goals of the project.

2.1 Game Rules

Objective: A castle has been invaded by evil skeletons armed with swords. It is the objective of the user to manoeuvre a knight to and through the castle, kill all the evil skeletons to finally face the Skeleton Lord. Underway it will be needed to solve small puzzles to further progress through the levels.

Non-player characters (NPC's): The game has two types of NPC's, which both are hostile agents toward the player.
The first type is a *Skeleton Warrior*-class armored with a sword. When the player is within a line of sight of 2 units, the skeleton will pursue and attack the player dealing 25 in damage per second. This class only has 50 health points (HP). The player can outrun these skeleton, whom will abandon its pursue of the player and return to its original guarding point.
The second type is a Skeleton Lord-class also armored with a sword, dealing 50 damage per 5 seconds, and has the ability to spawn Skeleton Warriors. This class has 300 health points.

Player character (PC): The player and main character, a swordsman, is controlled by the user. With the keyboard-keys: W, S, A, D, the user can move the player up, down, right and left, respectively. With key F the player will attack with its sword, if an enemy or other attackable objects are within the players line of sight. The player deals 25 damage per second and has starts with 200 health points. If needed, the player can find small health-potions around the map, which restores the players health to 100%. Furthermore the player has the ability to jump by pressing space. The player has five lives to begin with. Whenever the players health points reaches zero, he looses a life.

Hazards: The game has a few hazardous objects and environments. Spikes are placed on ground level, which deals 15 damage to the player on collision. Some places have lava streams which will bring instant death to the player. The only way to avoid damage from hazardous objects are to go around or jump over them. The lava-stream can be cooled down to icy conditions, hence the player will be able to pass over the frozen lava, by placing large rocks on top of trigger.

Level exits: To complete a level all enemies must be killed. First then, the exit-portal to the next level will be spawned. To enter an exit-portal simply jump or walk into it. On some levels this will be harder than others.

Potions: As mentioned earlier, the player can pick up health potions. This is done by walking through the potion, which immediately heals the player.

Game Over: If the player loses all their life, the game ends and all progress up to this point will be lost.

Movable objects: Large rocks can be moved by the player to aid the them in solving puzzles throughout the map. To move a rock the player must be adjacent to it and simply move in the desired direction.

2.2 Desired Features

Attack Variety: The game genre Hack-and-Slash have a large focus on combat and fighting. This is usually seen in having a large and varied move set for both the player characters and enemies. Having the ability to attack with either a heavy or light attack and following each of these up with various combinations of the two options, would be the desired version of combat seen in the final version of the game.

Cooperative Multiplayer: Having the ability to play the levels with one or more friends adds a lot to a Hack-and-Slash game. Multiplayer was however outside the scope of this course, which did change how levels and balancing was done in the end.

2.3 Technical Challenges

As described, our inspiration comes from the 2.5D side-scrolling Hack-and-Slash game Castle Crashers, where 2.5D refers to a game which is 2D but allows for Z-axis movement of sprites. In order to developing such a perspective, this group had many ideas and discussed various ways to implement such a solution. The first being implementing the whole project in a 2D-world and using 2D sprite's vertical movement to give the illusion of a third axis. However, after much experimentation and needing to allow the player to jump, this was changed in favor of creating a 3D-world, then restricting the camera to follow the player on the X- and Z-axis only, thus giving the illusion of a 2D-side scrolling game where the player can move (in a restricted range) along the z-axis as well. All sprites are still in 2D, which yielded a somewhat funny looking perspective, when the player was moving. But due to the camera being leveled in the beginning, when the player executed a jump, the player-character would jump toward the camera, instead of what would be considered straight up in the camera's perspective. Besides this, the view of the playing field felt narrow and it was hard as a player to get a feel for the complete overview of the playable area. To fix this, we rotated the camera by 11°downwards, and moved the camera up higher. This solved the visibility issue, but the 2D sprites in the scene still looked wrong and the jumping-issue persisted. By rotating all sprites, 11°on the x-axis, this includes ground, background 'sky-box' (the mountains), walls (in front and back), in-game items, such as boxes, stones, flags, and also enemies, everything was looking perfect. However, a rotated ground was not a good idea. This would have an impact on the gravity of non-static objects, like the player and enemies, being slowly pulled by gravity, giving them a sliding movement towards the camera. The solution was to make the ground have no rotation which, surprisingly, still looked very good. With a lot of experimentation and luck, a working 2.5D scene was implemented.

3 Implementation

3.1 Technical design

The game's levels are separated into their own scenes, this includes the Main Menu, which is actually a duplicate scene of the first level, where player and enemies are just disabled, and a UI canvas has been added to make the player choose between starting the first level or trying out the tutorial scene.

To handle player movements, CrossPlatformInput from Unity Standard Assets was imported to the project. This makes it easier to handle code to multiple platforms, such as PCs that uses a keyboard or a gaming console, such as a PlayStation (This is explained in greater details in the script-section).

All assets used in the game are placed in their appropriate folders. On the top-level, the folders are:

Animations, which contains the animation made from joining multiple sprites in sequence.

Download, are all content downloaded from Unity Assets Store or other sources, meaning content we did not create our self.

Plugins, contains a GitHub-version control plugin.

Prefabs, are template objects from which we can create new instances in all scenes, such as the player-character, enemies, the game manager, potions, all which are needed in more than one scene.

Scenes, contains all levels, including the main menu, a win and game-over scene.

Sprites, contains all 2D images used in the game. They are separated into sub folders based on their usage. Some sprites are large images with multiple sub images, and are thus, with the sprite editor, cut into smaller sections. Textures and Materials are also located in the sprites folder. These are a mix of original content and free assets.

Terrain, is the last folder and contains the terrain model. The terrain allows to create and sculpt a landscape, much like a drawing tool. This allows to quickly create landscapes and paint directly on them with textures. The terrain is used as the ground in each level of the game, which the player and enemies can move along, and sprite assets are placed upon. One very useful feature of the terrain model, is that it allows for navigation AI (artificial intelligence), which makes it easy make the enemies utilize path-finding which is used to chase the player (this is explained in greater details in the scripts-section).

Scripts, contains all scripts to handle the logic of the game, and will be elaborated in greater details in the following section.

3.2 Scripts

PlayerController.cs: Overall, this script handles how the player can interact, move, attack and discover objects in the game.

Run(): This function handles the movements of the player.

CrossPlatFormInputManager: This function allows for the group to simply focus on programming that works for multiple platforms. The scripts is attached as component to the player object, which is required in order to tap into the player's rigid body and add an velocity to the player based on the speed settings.

takeDamage(dmg): This function handles damage dealt to the player from enemies or hazardous objects. Furthermore this function calls the Game Manager to update its health point-bar.

RayCast(): This function is used by the player to be able to know what it is looking at, which is done by using raycasting in order to detect what is in front of the player. If the ray hits (detects) anything, the global Transform, *target*, will be assigned this hit, and it will set the global boolean, *canAttack*, that allows the player to attack.

Attack(): If the global *target* is equal to the tag *Enemy*, the player deals damage to the target, by tapping into the targets *Health*-component (which will be elaborated on later).

FlipSprite(): In order to make the player-sprite face the direction of the movement, the sprite must be flipped in that direction. This is done by transforming the local scale of the player sprites X-axis.

Besides these mentioned functions, several functions have been implemented to handle jumping and dying.

EnemyController.cs: Much of the same logic is applied to the *enemyController* as the *PlayerController.cs*, however the enemy is not being controlled by the user, instead an *MoveEnemy(Vector3 dest)*-function handles the logic of chasing the player or a taunt-point (which will be explained in EnemyTaunt.cs). This is done by creating a *NavMeshAgent* derived from the Unity.AI library.

EnemyTaunt.cs: When a object has this script attached, it can become a taunt-point for the enemy characters. This means, whenever the player is out of the enemy's line of sight, the enemy will seek back to the taunt-point. This is done by tapping into the enemy's *NavMeshAgent*-component and setting its destination.

AI.cs: This script handles the AI logic of the enemy in regards to coordination with *EnemyController*-script. It is responsible for finding the player, checking that the player is alive, in line of sight and in attack range . If all of these are true, the enemy will attack the player, which is handled by the *EnemyController.* To prevent the enemy and player sprites crash into the exact same spot (coordinates), *stoppingRange* will make the enemy stop within 1.2 units of the player. This script could actually be merged with the *EnemyController*, since it, in reality is more of a helper-script, rather than an AI-script.

Hazards.cs: This is a very simple script that only functions as a trigger-event. When an object has the script attached as a component, the player will die instantly on collision with that object.

Health.cs: When objects or enemies have this script attached as a component, they become a kill-able target for the player.

TakePotion.cs: This is a simple script, which when attached to a object they become a healing-potion. The script uses a trigger-event to detect when the player collides with the object, which then restores the players health to 100% and updates the health bar-UI.

LavaToIce.cs: When objects have this script attached, they become a trigger-event. Whenever the player or the ball-object is colliding with the object the lave-stream will turn into ice, when the player or ball is removed from the object again, the ice will turn back into lava. This scripts takes the object, which needs the lava /ice-conversion, as argument (Serialized Field) and two textures as arguments as well.

LevelManager.cs: This script is responsible for keeping track of the current scene, such that when the script is triggered, the next scene can be loaded. When an object in the game has the script attached it becomes the trigger.

Menu.cs: is another simple script that is attached to the *Main Menu*-UI buttons. Each function is responsible to load their respective scenes, and these functions are tied to their respective buttons in the UI. For instance *Start Game* will launch scene 1, which is the first playable level.

3.3 Design reasons

When discussing how the game should be designed and implemented, several different ways to be able to do this was identified. One of these was creating a single script that exclusively handled movements, for both the player, the enemy and objects. However, it was concluded that this would complicate things more, as the logic would handle both movement, attacking, health, path-finding, etc for all elements and objects with the same script attached. It was therefore decided to split the logic into separate scripts, one for each type of element. This results in having a lot more scripts, but also having clear and more manageable code which is way easier to debug.

For the *GameManager.cs* script the desire was to, besides managing the HUD (heads-up display) which displays the current health-point amount, have a way to display current amount of lives. After implementing the first solution to this, it was discovered that the data would not persist when a new level is loaded. This lead to discovering *singletons*, where the whole script *GameManager.cs* is treated as a session, such that, if a new level is loaded, and an session already exists, the new session would be destroyed, and the old one would persist, and thus continueing to hold the data of the previous level. This is handled in the *Awake()*-function of the Game manager.

3.4 Implementation issues

One of the major issues is to make a more complex enemy path-finding. Right now the enemy-characters are following the path-finding of the terrain model. The issue in our particular case is, for instance on the tutorial level, that enemy-agents cannot cross over the lava-stream (this is fine), but when the player activates the trigger and turns the lava-stream into ice, and thus the player is able to cross over the ice, the enemy-agents should also be able to do this. This is not possible because the lava-stream / ice objects are not part of the terrain, and thus, the terrain-model cannot create a path there. This yields some unwanted behaviours from the enemy-agents. We did not, in time, figure out, if the terrain model has a workaround to this issue, or we should implement a path-finding algorithm our self.

Minor issues are getting animation speeds just right, since some was original content while others were free assets. Changing the sampling rate did help getting these closer to each other, but there are still a mismatch with these.

4 Analysis

Having designed and implemented the prototype for *Knight Wars: Attack of the Bones*, the final version will be evaluated in regards to its playability, which is measured by the response received from play testers.

4.1 Iterations and Feedback

The implementation happened in stages, with the first being having a flat side-scrolling level, with a player-character. This version was used as more of a proof of concept for movement in a level.
Showcasing this early iteration was received well, with a desire for fine tuning the animations a bit, since the sprite's movement was a kind to floating around the level while flailing with its legs.

The next stage was implementing combat and enemies, which needed path-finding AI and attack scripts. The goals of this version was to be functional, giving the player the ability to hit and get hit.
The feedback for this iteration was less well received as the earlier one, due to players finding it hard to be in front of enemies. Players would swing their swords wildly without hitting anything, which meant the control were a bit off. In order to fix this, a large see-through cylinder was added to enemy sprites which inadvertently made collision better too.

Having the basic controls in place, the next version focused on level design and platforming. This version featured what ended up being level 1, with differences in terrain heights and a river of lava to jump over. A level exit was also added and locked, requiring the player to kill all enemies in the level before unlocking. The level exit lead to a placeholder level 2, which was not complete at this point.
Feedback to this first level was mixed, since the addition of the lava lake lead to some trouble for the testers. The idea was for the player to use a double jump over the river, which either due to how the double jump worked or the size of the lake, but players found this hard to do.
The idea of jumping over the lake was scraped in favor of having a puzzle. A pressure plate was added along with a rock which needed to be pushed unto the plate, which would then freeze over the lava lake. As a hint, if the player stands on top of the pressure plate, the lava lake would still freeze over.
This change was favored by the testers compared to the jumping puzzle, and the double jump was removed (normal jumping was still implemented). Players did figure out what to do remarkably well, which might be due to a push-able rocks and pressure plates being a well establish trope in gaming.

Moving unto creating the finale version, where the idea of having a boss fight was implemented. This meant creating a underground level and prefab for the boss, as well as having an attainable win-condition for the prototype.
Testing this iteration took some fine tuning with the help of the testers, since the boss should be too easy or too hard to overcome. The reaction to fighting the final version of

the boss was moderate, but this was mainly based on the simplicity of the combat system, with every tester saying they that it was fine but they missed having variety in the combat. Keeping this in mind the boss stayed was it was, with the plans of expanding the combat system further if time allowed (but time did not allow for this to be implemented). An original composition of the Italian folk song *Tarantella Napoletana* was added for background music.

5 Conclusion

Having implemented and tested the final iteration of the prototype for *Knight Wars: Attack of the Bones*, it is easy to see the simplification of the original pitch, but since this is a prototype that was to be expected. A game like Castle Crashers took a whole team to create, while this is merely a prototype of a game aiming to mimic it and the Hack-and-Slash game genre.

Combat was reduced to a single attack, while the focus was on having working hit detection. This would of course be a feature which needs to be expanded upon, due to this genre of game's reliance on this for the core gameplay. To make up for the combat's simplicity additional gameplay loops was looked into, with platforming being tried but scrapped in favor of a puzzle. If given the opportunity adding more levels and trying to implement platforming again would be ideal, since the genre and platforming are connected. Having an integrated tutorial level would also be something to be aimed for in the future.

If time allowed for it, creating original assets with the same style would also be ideal, since the mix of assets does result in a bit of visual dissonance. The problem with this is the time it takes to draw, rig and animate new enemies takes a lot of time, and creating textures which fit the aesthetic while still being visually pleasant and readable by the player also takes a lot of effort.

Overall, the resulting prototype for *Knight Wars: Attack of the Bones* has the feature core to the genre and is in a playable and acceptable state. Play-testers have responded positively to the final iteration, with the exception of their opinions of the combat system, but it works. Should work continue on the prototype, the resulting game would most likely be an okay Hack-and-Slash game.

6 References

6.1 Game Manual

Due to time constrains, having an in-game tutorial was scrapped, and a short manual of the controls will therefore be described here. When in a menu, the user's mouse is used to navigate and interact with the menu.

Movement: In order to move the player character, a user have to use *W*, *A*, *S* and *D* in order to move in the north, east, south and west respectively. These can be used in unison to move in a diagonal fashion, except when two opposite direction are used at the same time. Furthermore, the user can press *Space* to jump, which can be used in unison with any of the movement-keys.

Interact: When in a level the player character can interact with the environment and enemies. Due to animation limits *Interact* is always an attack, and such an attack is initiated by pressing *F*.

6.2 Technicalities, coding

https://wiki.unity3d.com/index.php/Singleton
https://forum.unity.com/
https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UICanvas.html

6.3 Sprites, images, art

https://docs.unity3d.com/Manual/class-InputManager.html
https://opengameart.org/content/ui-pieces
https://www.123rf.com/photo_39278572_seamless-ice-texture-abstract-winter-background.html
https://www.pinterest.se/pin/591660469771184150/?amp_client_id=CLIENT_ID
(_)mweb_unauth_id=amp_url=https%3A%2F%2Fwww.pinterest.se%2Famp%2Fpin
%2F591660469771184150%2Ffrom_amp_pin_page=true

https://assetstore.unity.com/packages/2d/environments/painted-hq-2d-forest-medieval-background-97738
https://assetstore.unity.com/packages/2d/environments/pixel-art-platformer-village-props-166114
https://www.freepngimg.com/png/27516-portal-photos

6.4 Games

https://en.wikipedia.org/wiki/Castle_Crashers
https://en.wikipedia.org/wiki/Streets_of_Rage
https://en.wikipedia.org/wiki/Dungeons_%26_Dragons:_Shadow_over_Mystara

7 Appendix

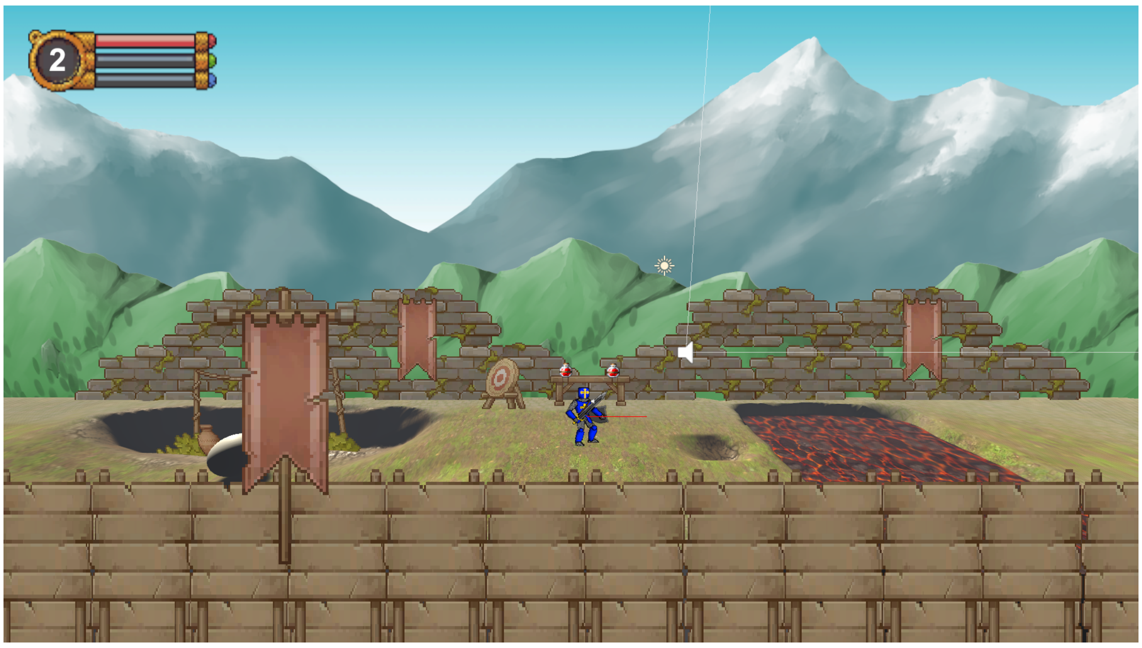

Figure 1: Screenshot of Level 1.

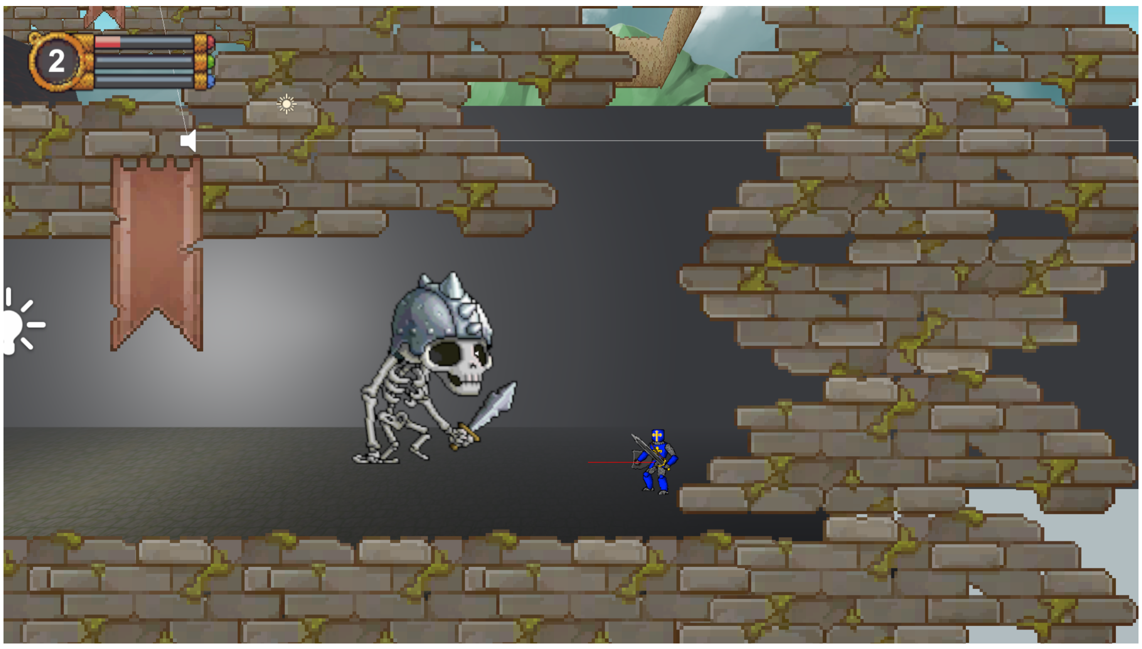

Figure 2: Screenshot of the boss in his underground lair.