Call of the Wintermoon

Kaylen Wheeler Arvand Dorgoly

March 12, 2012

Version History

Version Number	Edited By	Date	Comments
0-1	Kaylen	21/02/2012	Created document
0-2	Kaylen	24/02/2012	Started Game Overview
0-3	Kaylen	24/02/2012	Started Game objects and Logic
1-0	Kaylen, Arvand	12/03/2012	Completed all necessary sections

Contents

1 Game Overview

1.1 Game Summary

Play as the greatest black metal fanboy that has ever lived! Dig your way to the bottom of the cursed glacier to obtain the Blackblood Axe, the most brutal

instrument of terror ever created, to begin your worldwide reign of darkness and evil. What evils await within, and will you be evil enough to overcome them?

1.2 Platform

The primary target platform will be Windows PC and Macs. Development for the PC/Mac has few barriesr compared to other platforms. Additionally, the game will be developed with relatively low technical requirements in mind, and this will open up the market to a greater number of devices. Many people own PC's, and developing for this platform will make the game very widely available.

Because this game is built with the Unity engine, further platforms, such as Xbox Live Arcade or Playstation Network may be considered at a later date, as the engine allows games to be easily ported to many platforms. However, desktop platforms are our primary concern.

2 Development Overview

2.1 Development Team

The development team consists of Kaylen Wheeler and Arvand Dorgoly. This project includes many different tasks such as designing levels, game scenes, game play, concept art, animation, modeling and coding. In almost all tasks, both team members take part. In different stages of the design process, if the team feels an important feature needs to be tested, they will code a simple prototype. Currently, all coding is in the prototype phase.

2.2 Development Environment

The Environment chosen for development was the free version of Unity engine. Unity can build games for many platforms, including Mac OSX, windows, web browser, iPhone, iPad, android phones, Wii, ps3 and Xbox 360.

The Unity engine itself doesnt need any special hardware requirements for execution. It can run either on Windows (XP or later) or Mac OS X(10.5 or later). Any 64 MB graphics card and pixel shaders or 4 texture units can handle Unity graphics. The development team has decided to work in a Windows environment because it is more easily accessible. Furthermore all the other development tools in this project such as Blender, Dia, and Git are compatible with windows.

The first version of the game will be released for PC. However, due to powerful features of Unity, porting the game to any other console requires only small alternations.

2.2.1 Development Hardware

No special hardware is required. Standard Windows 7 PC's are being used for development.

2.2.2 Development Software

Dia Dia is a free drawing tool for Windows (XP or later) and also Mac OS. All the diagrams in this document including activity, UML, FSM and flow charts were drawn with Dia. It can easily export diagrams to .tex format which is quite useful because the design document has been written in Latex.

teXworks TeXWorks is an open source application for Windows, Linux and Mac OS. It is a text editor with a graphical user interface that allows quick compilation of latex documents to PDF format.

Blender Blender is a free open source 3D modeling and animation tool. All game objects (apart from Unity's primitive meshes) such as enemies and player avatar models will be modeled and animated in Blender. Importing these models to Unity doesnt require any plugins. All Blender assets are automatically imported if placed in the Unity project folder.

Unity Engine We are using free version of Unity 3.4 which is downloadable from Unity's official site. This engine has lots of built in functionality and objects which makes it easy to work with. The GUI design in the Unity editor is quite straightforward. The ease of use, combined with very low script compilation times, enables the development team to check design decisions rapidly by generating prototypes. Consequently it leads to better design choices. The fact that porting the game to any console other than PC takes minimal effort was another important factor influencing the decision to use this engine.

MonoDevelop Mono is an open source implementation of Microsoft's .NET framework. MonoDevelop is the primary IDE used with Mono. By default, Unity includes a specialized version of MonoDevelop. In Unity there are three options for scripting languages: UnityScript (Similar to JavaScript), C# and Boo(a strongly-typed, Python-inspired .NET language). The development team decided to write scripts with C# because all team members have adequate experience with this language.

Git The Git version control system is used to manage changes to code and other assets. Code is currently hosted on GitHub (using the free license), a swb-based hosting service for software development projects.

2.2.3 External Code

The code used for level generation with the Growing Tree algorithm was translated into C# from a Python source file found in the following article from the procedural content generation wiki : `http://pcg.wikidot.com/pcg-algorithm:maze`.

We are considering using a metaball rendering technique to simulate the amorphous nature of the Shoggoth enemies. C# code for creating this effect was obtained from `http://www.unifycommunity.com/wiki/index.php?title=MetaBalls`.

Some of the code related to mouse pointing and aiming was take from a Unity example project, which can be found at `http://unity3d.com/support/resources/unity-extensions/head-look-controller.html`.

3 Game mechanics

3.1 Main Technical Requirements

The Unity website suggests the following minimum system requirements (`http://unity3d.com/unity/system-requirements.html`).

- Windows XP or later; Mac OS X 10.5 or later.
- Pretty much any 3D graphics card, depending on complexity.
- Online games run on all browsers, including IE, Firefox, Safari, and Chrome, among others.

It is clear from this list that techical requirements vary greatly depending on the specific game implemented. Because graphical complexity is not a core aspect of our game, we will make attempt to make technical requirements as low as possible.

3.2 Architecture

Since the Unity engine is being used to create and run this game, software architecture is heavily influenced by the architecture of the engine. However, there are still some important aspects of software architecture that must be decided on a per-game basis. This section will outline both Unity's software architecture and how we used that architecture to implement that of our own game.

3.2.1 Overview of Unity

Unity is an integrated authoring tool used for creating 3D games. By default, it provides many of the common technical elements present in most 3D games, such as high-performance graphics and physics. Graphics are implemented using either Direct3D or OpenGL, and Unity also supports Nvidia's PhysX engine. Implementation of each of these systems varies depending on the platform of deployment.

While the core of the engine has varying native implementations on different devices, scripting is implemented using Mono, the open-source implementation of the .NET Framework. Mono provides a software framework that allows code

to run on multiple platforms. Code is compiled to the Common Language Runtime (CLR) so that code written in different high-level languages can interoperated. In Unity, three different languages are supported - UnityScript(Similar to JavaScript), Boo (Similar to Python) and C#. Because of the development team's experience, C# was chosen as the primary scripting language.

To implement scripted behaviour, the MonoBehaviour class is used. This class is part of the UnityEngine library. Multiple MonoBehavour objects can be attached to any game object, and each of them adds some behaviour to that object. The class provides several hook methods, most important of which is Update, which is called once per engine update cycle (update cycles vary, but occur approximately 60 times per second). In addition to update functions, several event-handling functions are also provided, which can be used to detect collisions or other events that may affect the object.

3.2.2 Our Architecture

Using the architecture already created by Unity, our own architecture will be created to suit the purposes of our game. The primary goals of this will be to handle the enemy AI and the combat system, which are specific to this game.

The classes involved in AI control of game entities are derived from the EntityControl class, which is itself derived from MonoBehaviour. See Fig 1 for a UML diagram depicting EntityControl and other classes. EntityControl's structure is based on that of a Finite State Machine. Each instance of EntityControl contains an array of Action objects which can be indexed by any member of the EntityState enumeration. Action is a C# delegate type (similar to a function pointer) representing a void function with no parameters.

3.3 Game Flow

The flow of control in code as it relates to the AI and Combat systems will be described in this section. The classes and systems referred to in thie section were introduced in the previous section.

3.3.1 State Machines and AI

During the Start function (a function of MonoBehaviour that is run when a game object is initialized), the StateFunctions array is automatically initialized to default values. During this stage, some indices of the array are matched up with their default functions (those marked as "State Function" in the UML diagram). These functions represent states that tend to be universal across all entities. Some indices of the array correspond to more specific states, and are therefore not initialized. This is done so that exceptions will be thrown for invalid states, allowing bugs in code to be caught more effectively.

The Update function in EntityControl functions primarily as a state-function dispatcher. Rather than use the Update function directly in the subclasses, hooks are provided to allow more organization. Flow of control is illustrated

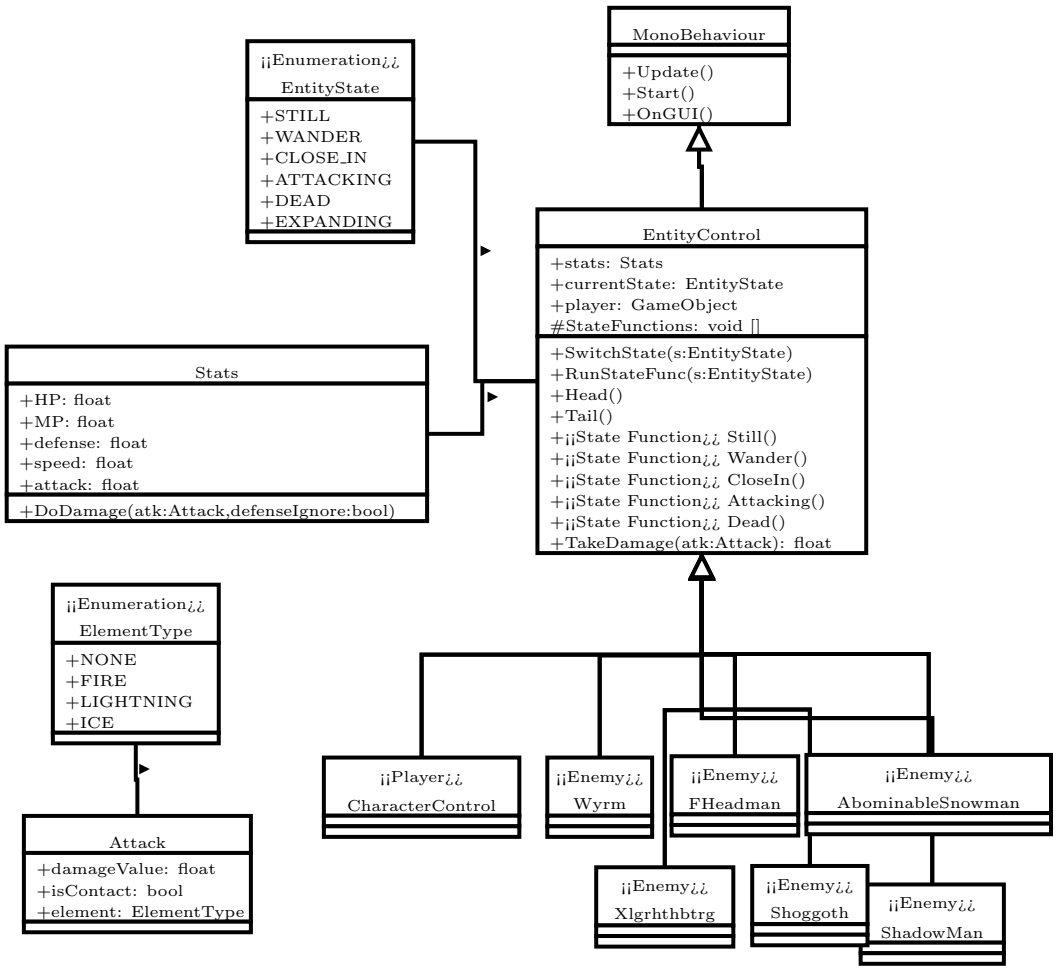


Figure 1: A UML diagram of the entity control and combat system.

in Fig 2. First, the Head function is called to execute any code that may be required before the state function. Next, the current state function is fetched from the array. If there is no function for the current state, and exception is thrown and execution stops. Otherwise, the current state function is executed. Control continues to a call to the Tail method, executing any code that may be required at the end of the update cycle.

3.3.2 Combat System

The combat system has its own complexities as well. Entities are not simply destroyed when attacked, but must process the attack to compute the final

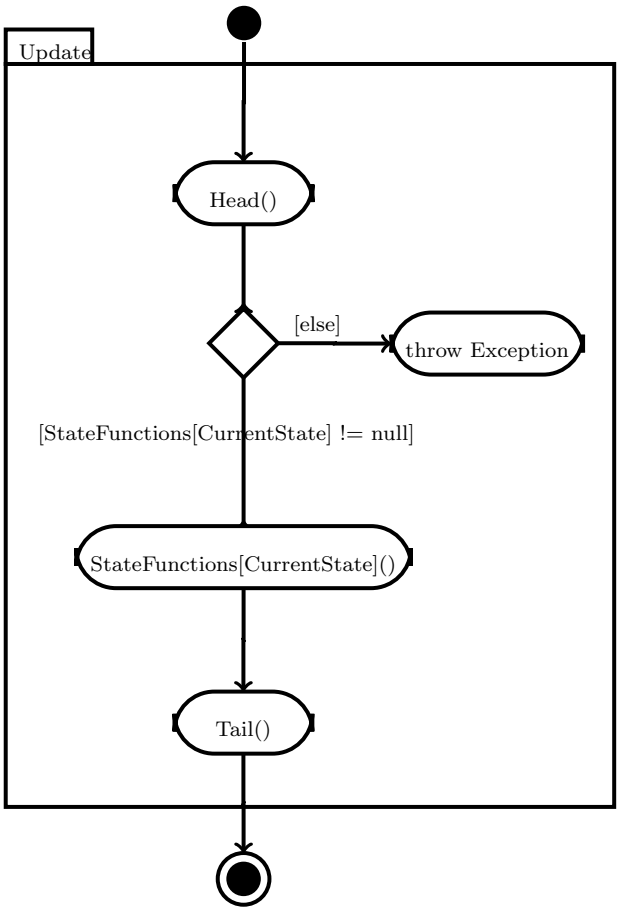


Figure 2: A UML activity diagram representing the update cycle.

amount of damage (see section 3.8.2 for more details on combat) . To store their HP, defense, and other attributes, an instance of the Stats class is used. To represent attack actions, the Attack class is used. This class represents all attributes of the attack (i.e. damage, element type, etc.) A sequence diagram representing combat interactions is displayed in Fig. 3.

Typically, the combat system functions as follows:

1. An object collides with an Entity.
2. An attack message is sent to the Entity.
3. The entity processes the attack message.
4. The attack message is then passed off to the Stats object.

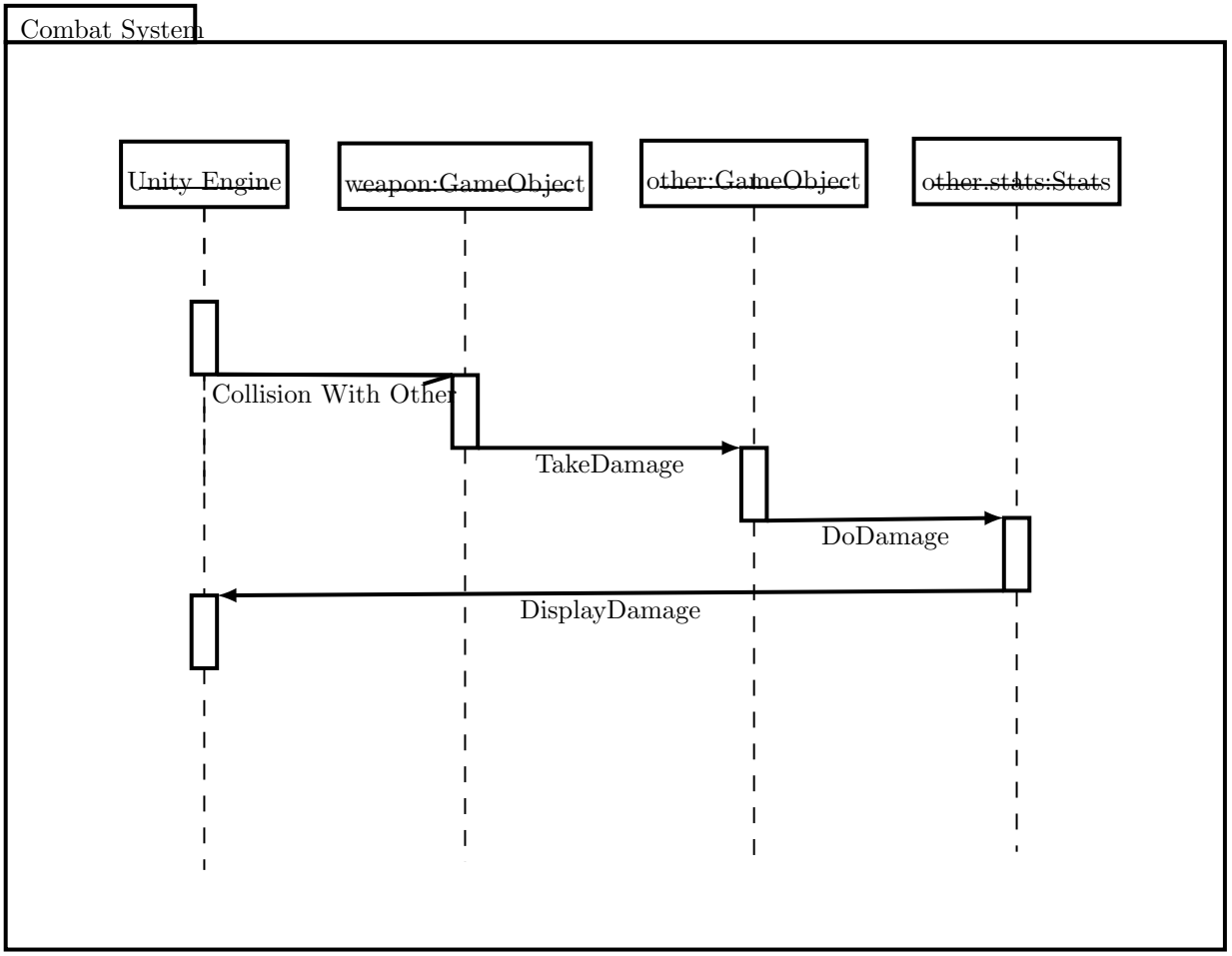


Figure 3: A sequence diagram showing combat interactions.

5. The stats object affects the necessary changes and displays the damage value on screen.

In order to more easily understand the process, a visual example will be shown. In this example, the player attacks an enemy. The white capsule-like object is the player, and the black sphere is an enemy. The red sphere is used to indicate the mouse position, and the green cube is used to indicate the hit-box associated with the attack action. The stats of each object are shown in Fig. 4.

Figure 5 shows the state of the world before attacking has started. When the player hits the attack button, the attack animation begins. This causes the hitbox to move forward and make contact with the enemy (Fig. 6). When the hitbox makes contact with the enemy, its collision handling method is executed

	HP	MP	Defense	Attack	Speed
Player	100	0	0	20	10
Enemy	100	0	10	0	10

Figure 4: Player and enemy stats.

by the game engine.

Figure 7 shows the code for the collision handling method. If the object that it has collided with is not the hitbox's owner (the player), and it has an EntityControl script, then a new Attack object is constructed and sent to that EntityControl script. The Attack object gets constructed and has its attributes set on lines 11-15. It can be clearly seen that the attack power is equal to that in the player's Stats object. The isContact flag is set to false, indicating that this is not contact damage, and the attack's elemental attribute is set to ice.

This Attack message is then handled by the enemy's TakeDamage method, which can be seen in Fig. 8. The Attack object is processed before the final damage value is sent to the Stats object. This particular TakeDamage method doubles damage if its element type is ice. The damage value is then sent to the Stats object, which subtracts the defense value from the damage before applying it to the HP.

The complete process looks something like this:

1. Hit box collides with enemy, collision handler is called.
2. An attack message with properties {attack=20, isContact = false, element = ICE} is sent to the enemy.
3. The entity processes the attack message, multiplying the damage by 2 (to get 40) because of the ICE attribute.
4. The attack message is then passed off to the Stats object, which subtracts the defense value (10) from the damage (yielding 30), and subtracts it form the HP value (yielding 70 HP).
5. The final damage value (30) is displayed on screen above the enemy.

3.4 Graphics

Despite having gameplay restricted along a 2D plane, the game's graphics will be entirely 3D. Models and animations will be created using Blender, and imported to Unity. The technical aspects of the rendering will be handled by Unity's graphics engine.

3.4.1 Unity Graphics Overview

Being a multi-platform engine, the specifics of the graphics engine vary between platforms. The two platforms we will be focusing on are what is referred to in

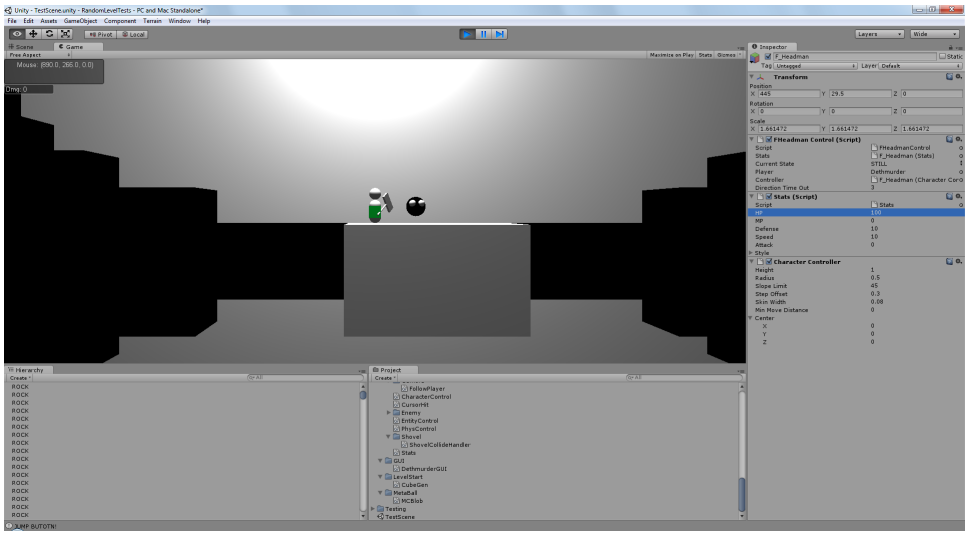


Figure 5: Player prior to attacking the enemy.

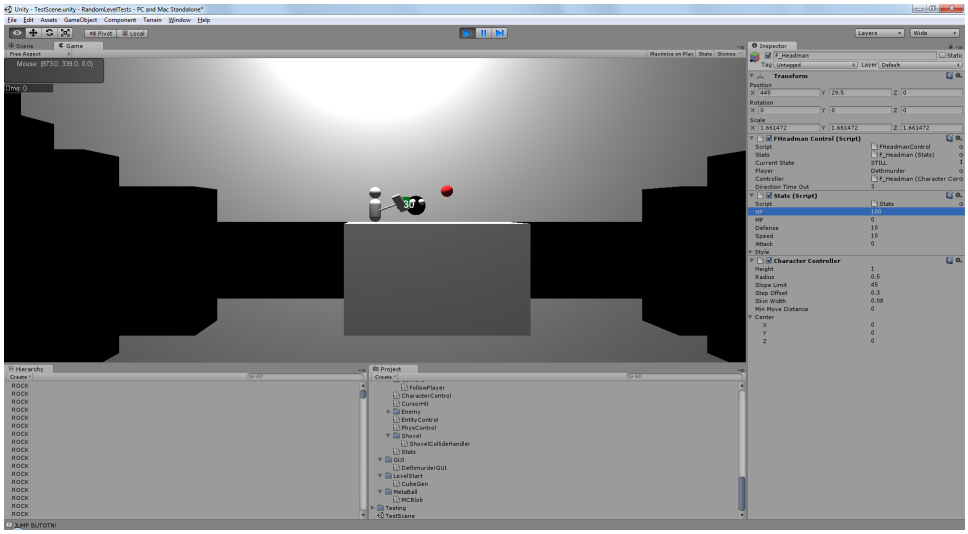


Figure 6: Player while attacking the enemy.

the Unity documentation as "Desktop" platforms, which includes Windows PC and Mac systems. The windows version of the rendering engine uses Direct3D, while OpenGL is used in the Mac version. For desktop games, a wide variety of rendering options are available, including over 100 shaders, the ability to create custom CG and GLSL shaders, and many more. Additionally, many rendering optimizations are present, such as deferred rendering and batching.

```
1  void OnTriggerEnter (Collider other){
2
3          if (other.gameObject != owner && characterControl
               .IsAttacking()){
4
5                  // Attempt to get an entity control
                       object from the other entity
6                  EntityControl otherControl = other.
                       gameObject.GetComponent(typeof(
                       EntityControl)) as EntityControl;
7
8                  // If we did get one, let's do damage to
                       it
9                  if (otherControl != null){
10                         otherControl.TakeDamage(
11                         new Attack{
12                                 damageValue = ec.stats.
                                       attack,
13                                 isContact=false,
14                                 element=ElementType.ICE}
15                                 );
16                 }
17         }
18 }
```

Figure 7: The hitbox's collision handling method.

```
1  public override float TakeDamage (Attack atk)
2  {
3          float dmg = atk.damageValue;
4
5          // This guy is weak against ice
6          if (atk.element == ElementType.ICE){
7                  dmg *= 2;
8          }
9
10         stats.DoDamage(dmg, false);
11
12         return dmg;
13 }
```

Figure 8: The enemy's TakeDamage method.

While the specifics of the rendering engine are much too complex to understand for such a small development team with limited time and resources, certain key details are revealed on Unity's website so that developers can optimize their graphics performance. The most significant of these details as they apply to our game are those about optimizing meshes. It is suggested that each mesh should be between 1500 and 4000 triangles, and that each should use a single material, as multi-material meshes are rendered once for each material attached to them. We will attempt to follow these guidelines as closely as possible to optimize graphics performance.

Additionally, there are several emulation modes available with the unity graphics engine, which can be used to test compatibility with older hardware. Because we do not want our game to have high technical requirements, we will make use of this feature.

3.4.2 Models and Animations

As mentioned, Blender will be used to create models and animation. The development team has experience with Blender, so there is an extremely low learning curve. Additionally, Unity includes built-in support for importing blender models and animations, further reducing the effort required to create 3D graphics.

Both Blender and Unity support skeletal animation systems, which further simplifies the task of animation. The positions of bones can be keyframed in Blender to produce animations, and those animations will carry over to Unity when imported. An issue that causes some concern is the absence of an inverse kinematics system in Unity, despite its availability in Blender. Inverse kinematics solvers enable chains of bones (such as limbs) to be more easily animated. However, by keyframing the position and rotation attributes of the bones controlled by inverse kinematics in blender, and through use of animation blending, we should be able to accomplish most animation tasks.

Animation blending is also supported by both systems, which means that muiltiple animations can be combined in new and interesting ways. One of the features this is used to implement is the player's directional attacking ability. Figures 9, 10 and 11 show how this can be accomplished. Only two animations have been defined manually - the forward hit and upward hit animations. Attacks always occur in the direction of the mouse pointer. In these cases, weights are assigned to the animations based on the angle between the player and the mouse pointer. The upward hit is proportional to the sine (dy / hyp), and the forward hit is proportional to the cosine (dx / hyp). Thus, when dx = 0, the sine is 1 and the cosine is 0, resulting in a complete upward hit. When dy = 0, the cosine is 1, resulting in a complete forward hit. For any angles in between, the weights are appropriately adjusted to combine the animations in the right proportions.

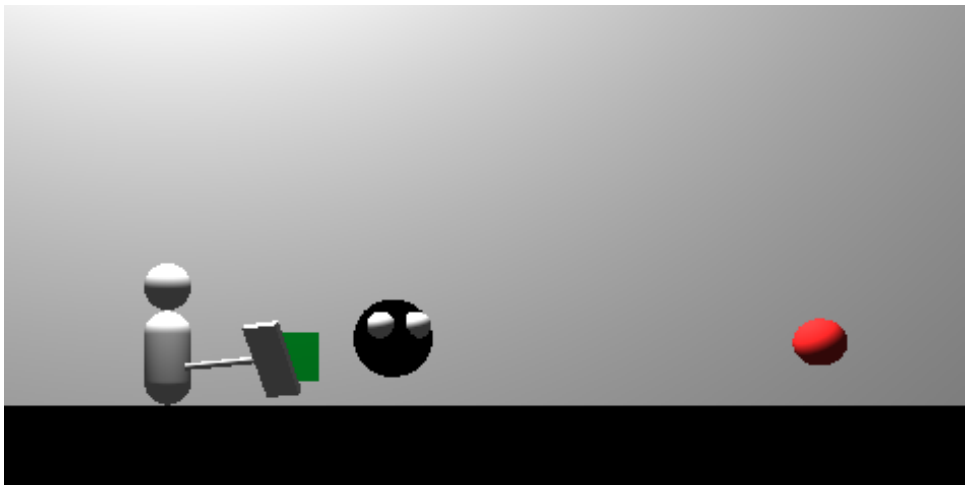


Figure 9: The forward hit animation.

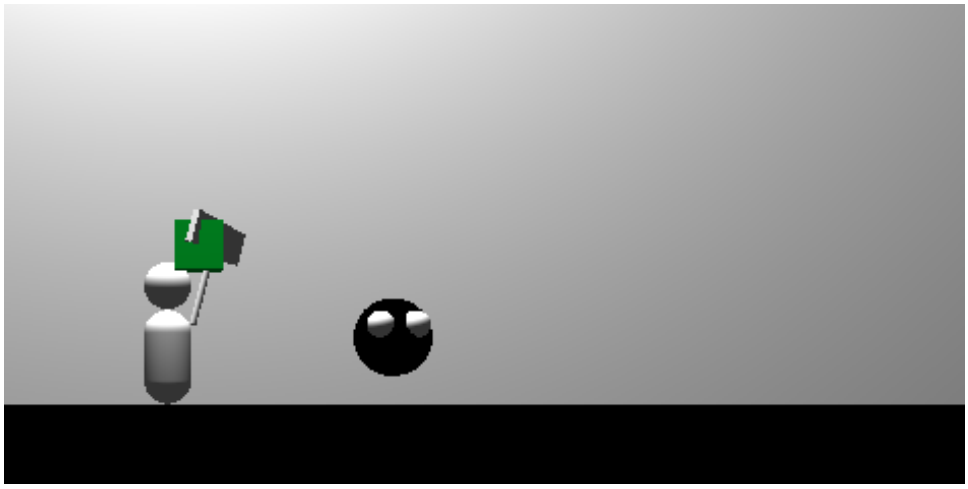


Figure 10: The upward hit animation.

3.5 Audio

Unity's audio system is very easy to use and supports a number of different audio types. Audio assets can be imported from .aif, .wav, .mp3, and .ogg formats. When the assets are imported, they can be built into one of 2 formats: native (.wav) and compressed (using Ogg Vorbis compression). Mono, stereo, and multi-channel audio (up to 8 channels) is are also supported.

Unity also supports 3D sound, allowing audio sources to be placed at 3D coordinates in the game world. AudioSource components are attached to game

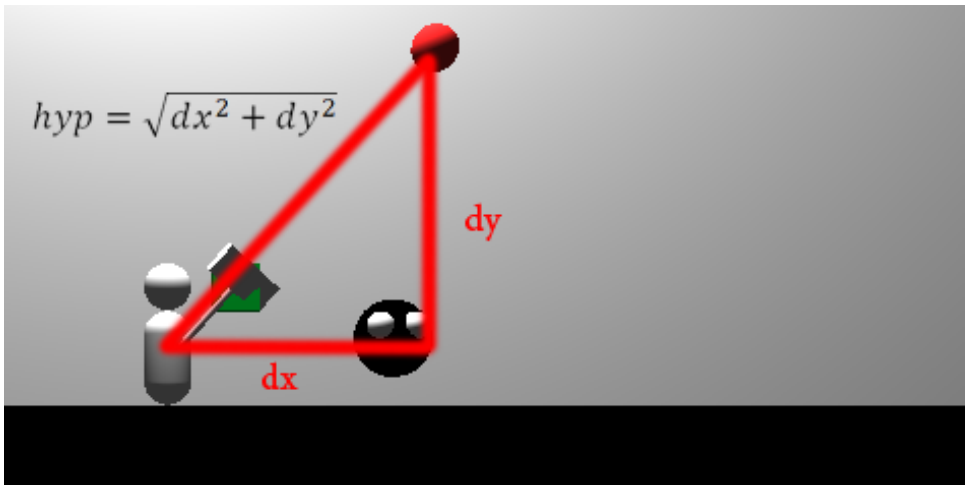


Figure 11: The result of combined up and forward hit animations.

objects, and will produce sound at their location. Many aspects of an AudioSource, such as reverberation and distance fall-off functions can be edited to suit the developer's needs.

3.6 Artificial Intelligence

In this game there will be different enemies with different attributes and qualities. Each enemy has a finite state machine which is managed by EntityControl class. Each enemy class overrides this class to get his own special behaviour, attacks and qualities.

As said before enemies are similar to the player in a sense that they all have stats. Each individual enemy object inherits from EntityControl. Inside of each entity controller there is a stats object. The functionality of the finite state machine model is outlined in section 3.1. In this section, the specifics of each enemy's finite state machine will be detailed.

3.6.1 Description of Common States

A number of states are shared among many of the enemies, and will have their own implementations in each enemy class. However, the core of their functionality is the same.

Wander The start state for almost all enemies is wandering. In this state the NPC moves around in random direction for a certain amount of time. He changes the direction after time out and chooses another random direction. For most enemies, this involves moving along the ground. Floating enemies such as F. Headman, Shadowman may wander in any direction.

Close In In each update the enemy checks for if the player is visible by ray casting. If the player is seen, the state machine will typically transition to close in state which the player is pursued. Pursuit stops chasing the player is in range, or when visibility is cut off. If the NPC faces an obstacle which hides the player, it will go back to the wandering state.

Attack If enemy successfully get to the range of the player, it starts periodic attacks. If the player gets out of the enemys range it will go back to the close in state.

Dead In each update the enemy game stats will be checked to see if his alive or not ,this means checking his HP level. If HP equals zero the enemy is dead. The dead state hasnt shown in any of diagrams but obviously in every update enemy HP will be checked and he can transfer from any of active states to dead one.

3.6.2 Enemy-specific Idiosyncracies

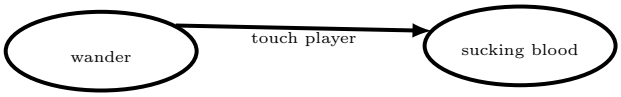


Figure 12: A Wyrm's state machine.

Wyrm The Wyrms lack a close-in state. Instead, they wander until they make contact with the player, at which point they latch onto the player and continue attacking. Wyrms are not highly sophistocated enemies. They have low HP and draw their strength from being in large groups.

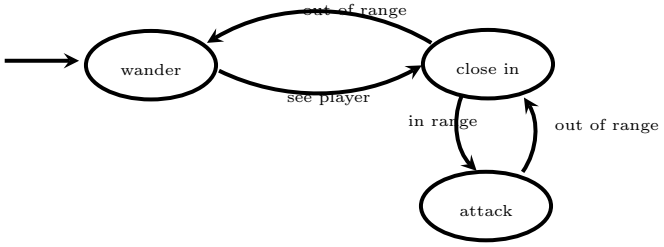


Figure 13: F. Headman's state machine.

F. Headman The F. Headman enemies follow the common states very closely. Its attack involves spitting fire at the player.

Shadowman The Shadowman is very similar to F. Headman. However, the attack is different. When it comes into close range of the player, it opens its cloak and expands. In this expanding state, it attempts to suck the player in. If the player is sucked in, it is attacked and expelled. If the player escapes, the Shadowman goes back to the wandering state.

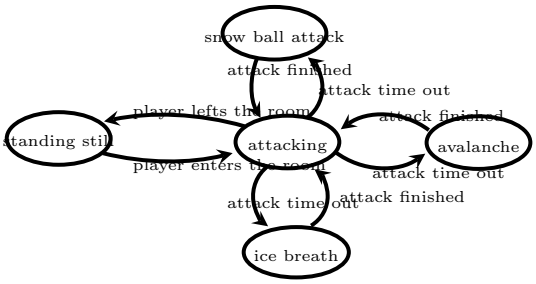


Figure 14: Abominable Shadowman's state machine.

Abominable Snowman The abominable snowman does not have a wander state, as it is a stationary enemy that relies primarily on ranged attacks. It automatically enters the attacking state from the idle state when the player enters the room, at which point it randomly selects an attack to use after a certain time-out. It continues this process indefinitely after the attack is complete. It has three options for attacks - throwing a snowball, using ice breath, and causing an avalanche from the ceiling of the room.

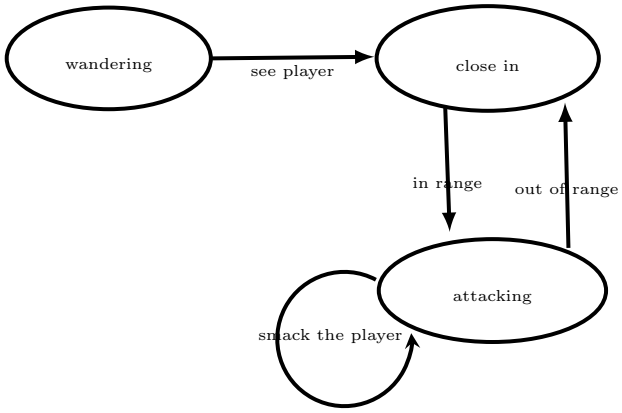


Figure 15: Shoggoth's state machine.

Shoggoth The shoggoths follow the common states very closely. Its attack involves hititng the player with its tentacle, which sends the player flying.

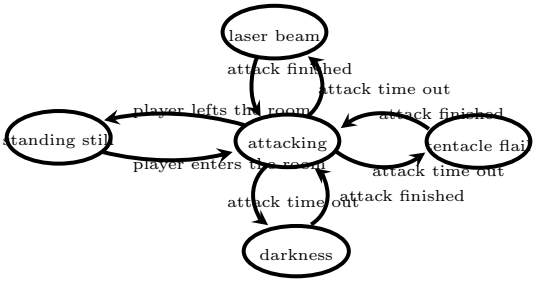


Figure 16: Xl'Grhthbtrg's state machine.

Xl'Grhthbtrg Xl'Grhthbtrg acts similar to the abominable snowman. Because it is a boss, it will be the only enemy in the room with the player during the battle. The only difference with the snowman is in the behaviour associated with the attacks themselves. It has three options for attacks - moving around while flailing tentacles, firing lasers from its tentacles, or filling the screen with "darkness", which stuns the player momentarily.

3.7 Physics

A physics engine is provided by default with Unity. Although our game will not make heavy use of the physics engine, it is important for handling collision detection. Unity provides a variety of basic collider components which can be attached to game objects. Among these are primitives such as cubes, spheres, and capsules. Colliders can be set as triggers, which enables them to detect collisions with other objects, but not affect those objects in any way (i.e. they can pass through other objects). The extent to whcih the physics system is used in our game will be primarily for collision detection and response, which can be easily accomplished through collision event handlers (see Fig. 7).

3.8 Game Objects and Logic

3.8.1 Gameplay Overview

Gameplay is restricted along the 2-dimensional XY plane, although the graphics are rendered in 3D. The basic player actions can be classified as moving, attacking, and jumping. There are several extensions of these actions that will be further elaborated upon in the Abilities section.

Levels will be randomly generated, and will consist of a 2-dimensional grid of cubes, some of which may be empty. The non-empty cubes will consist of a variety of different materials (i.e. snow, ice, rock, etc.). Attacking cubes made of snow will allow the player to further progress in the game.

As the player explores the game world and defeats enemies, they will gain experience points. The number of points gained is specific to the enemy type.

3.8.2 Player and Enemy Attirbutes

Players and enemies share certain attributes that determine how they act in combat. The specific rules of combat are detailed in section 3.8.2.

HP Determines how much damage an entity can take before it is destroyed. In the case of the player, HP is displayed in a number of discrete blocks. Each consists of 100 HP. (Similar to the energy tanks of the Metroid series.)

MP Depleted when the player uses special abilities. MP is also displayed as a number of discrete blocks, similar to HP.

Attack The amount of damage dealt by an attack.

Defense Resistance to attacks.

Speed Affects movement and attack speed.

3.8.3 Combat Mechanics

When an attack connects with an entity, the attack attempts to remove HP equal to its power. Before that damage is applied, the entity's defense value is subtracted. The resulting value is clamped to a minimum of 1 HP of damage.

3.8.4 Abilities

(Abilities that must be obtained after beginning the game are indicated by *.)

Moving Using directional keys (A and D by default), the player can move left and right. The direction the player faces is determined by the direction of the mouse pointer from the player, not the direction of movement.

Jumping Jumping is by default accomplished with the Space key. Holding the jump key longer results in a higher jump. The trajectory can be controlled in midair by using the directional keys.

Crouching Using the down button (S by default), the player can crouch in order to avoid enemy attacks. They are still able to attack in this state.

High Jump If the player hits the jump key when crouched, a high jump will be executed. This will allow the player to reach high areas.

Wall Jumping A wall-jump ability is also available. The ability can be used repeatedly in order to ascend walls.

Attacking and Digging The standard attack is a simple slash with the snow shovel. This will remain the same regardless of the level of the player, although the attack power may increase. Standard attacks can be aimed in any direction, according to the position of the mouse cursor.

If the attack hits a cube of ice, it will be destroyed.

Special Abilities* A number of special attacks will be available to the player. They will be able to use them if they find the necessary items. The attacks are listed below.

Lightning Bolt* A bolt of lightning is fired in the direction of the cursor. Lightning elemental attack.

Flame Strike* A powerful slash with the shovel, but the shovel is covered in flame. This results in greater damage and an increased area of effect. Fire elemental attack.

Ice Shield* A temporary shield of ice is generated around the player, stopping projectiles and damaging any enemies that come into contact with it. Ice elemental attack.

Heal* A special ability that allows the player to recover HP without using obtaining a health powerup.

Dash* Allows quick, short bursts of movement. Ideal for fast motion and evasion of enemies. The player can dash both forwards or backwards. The default dash keys are Q and E, used for left and right dashes respectively.

3.8.5 Enemies

Wyrm Millenia of exposure to the black Majyyks of the cursed glacier have turned worms into Wyrms! They have little wings and sharp teeth and are scary now! Watch out for their blood sucking attack. It will suck your blood!

F. Headman Question 1: Does the F stand for Floating or Flaming? Question 2: Does it really matter? F. Headman is one of a decapitated horde of fire-breathing black metal singers cursed to float around and breathe fire for eternity.... which is not really a curse because its totally awesome! Anyway, he can shoot fire at you, so watch it!

Shadowman A cursed soul destined to wander the bowels of the earth forever in search of flesh to satisfy its hunger. Hes a robe hooded guy with skeleton arms or something. He may seem fairly harmless, but he will open his robe to suck you in, so watch out!

Abominable Snowman Wow, look at this guy. He has a knife for a nose! That is not a carrot. THAT IS WHAT MUTILATES CARROTS! Just like this guy will mutilate YOU!

Shoggoth It was a terrible, indescribable thing vaster than any subway traina shapeless congeries of protoplasmic bubbles, faintly self-luminous, and with myriads of temporary eyes forming and un-forming as pustules of greenish light all over the tunnel-filling front that bore down upon us.

Figure 17: Wyrm

These were the last words of HP Lovecraft before he was eaten by one of these bad boys. You dont want to get in their way. Theyll form an un-form

Figure 18: F. Headman

pustules of PAIN on the inside of your BRAIN!

Xl'Grhthbtrg The power of Xl'grhthlbtrg is not one that can be known by the simple minds of men, for his very existence is drawn from the blackest voids beyond the abominable stars. There he beholds the unspeakable creations of the Nameless One and screams his master's dark will across the cosmos.

Also, hes the final boss of level 1. Youd think that with all those cosmic powers, hed be able to land a better gig. Guess not!

3.8.6 Levels

Levels are randomly generated with every new game. Level generation begins with the construction of a random maze, which is then processed to produce a complete level. Enemies are then added to the level. Level generation follows a three-step process, which is outlined here.

Maze Generation A maze generation algorithm called the Growing Tree algorithm is first initialized with a random seed, and proceeds to generate a

Figure 19: Shadowman

maze. this maze generation algorithm ensures the creation of a maze with no loops. That is, there is only one path between any two empty squares. An

Figure 20: Abominable Snowman

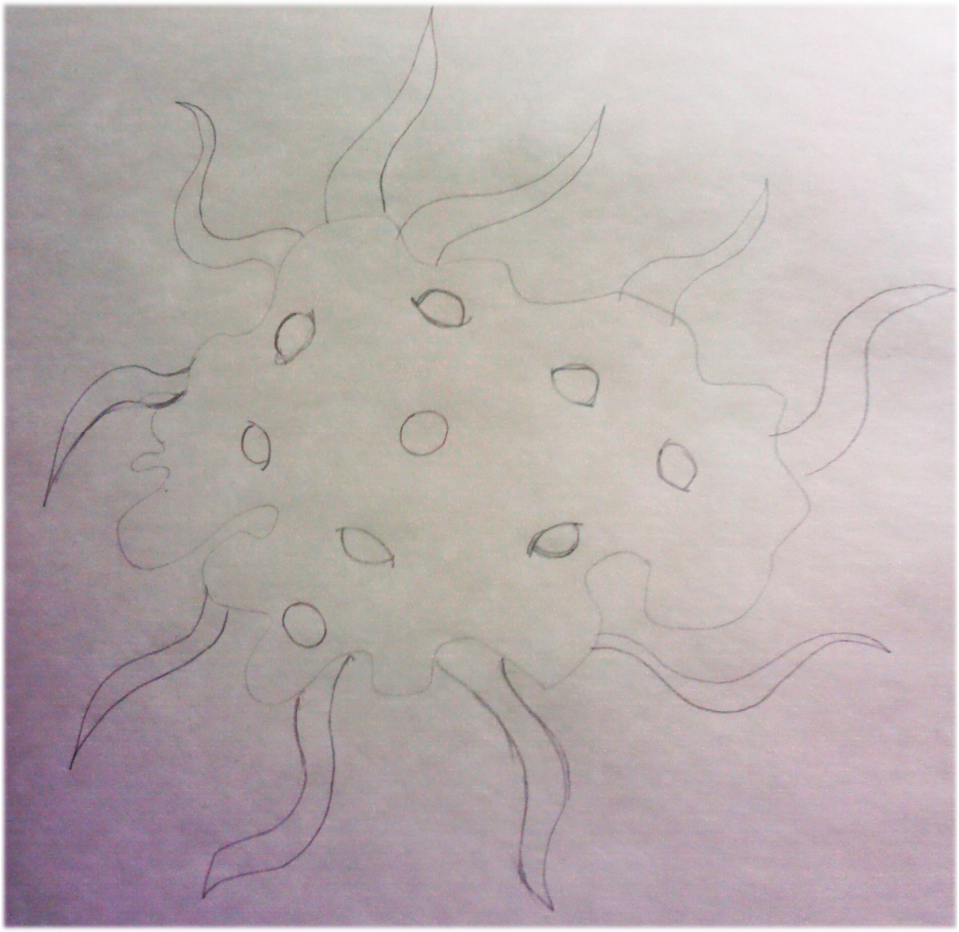


Figure 21: Shoggoth

example of such a maze can be seen in figure 23. In this figure, X' represent walls, while periods represent empty spaces.

Room Processing Once the maze is created, it is converted into a level. This is done by replacing each empty square with a randomly selected Room type. Room types are visible in section 6. Once the rooms are in place, a second pass over the grid checks for adjacencies between rooms, and replaces the walls, ceiling, or floor with snow in order to allow the player to pass between adjacent rooms.

Enemy Placement The third step involves placing enemies in the rooms. For each room type, a number of enemy configurations are available. These

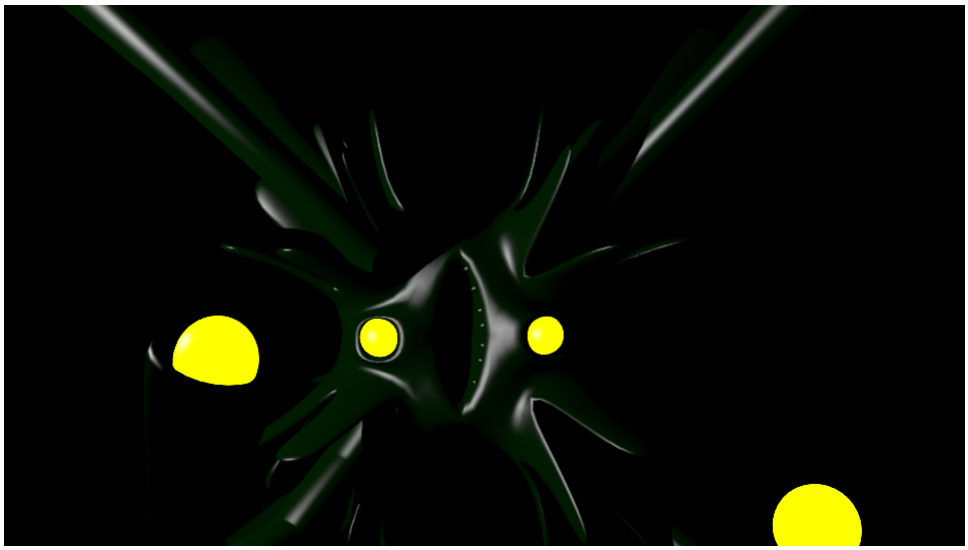


Figure 22: Xl'Grhthbtrg

configurations are visible in section 6. For each room, one of these configurations will be selected at random. One exception to this rule is the boss room, which may only appear once per level.

3.8.7 Gaining Experience and Abilities

Increasing player stats and gaining abilities is achieved through use of the level-up pentagram. The level-up pentagram (fig. 25) can be accessed through the status menu by pressing tab on the game screen. The level-up pentagram enables customization and specialization of stats and abilities, allowing different play experiences for different play-throughs of the game.

```
....XX...X
X.X....X..
...XX.X.X.
.X...X...X
.X.XX..X..
XX..X.X.X.
...X..X...
.XXX.X..XX
....XX.X..
.XX......X
```

Figure 23: A maze generated with the Growing Tree algorithm.

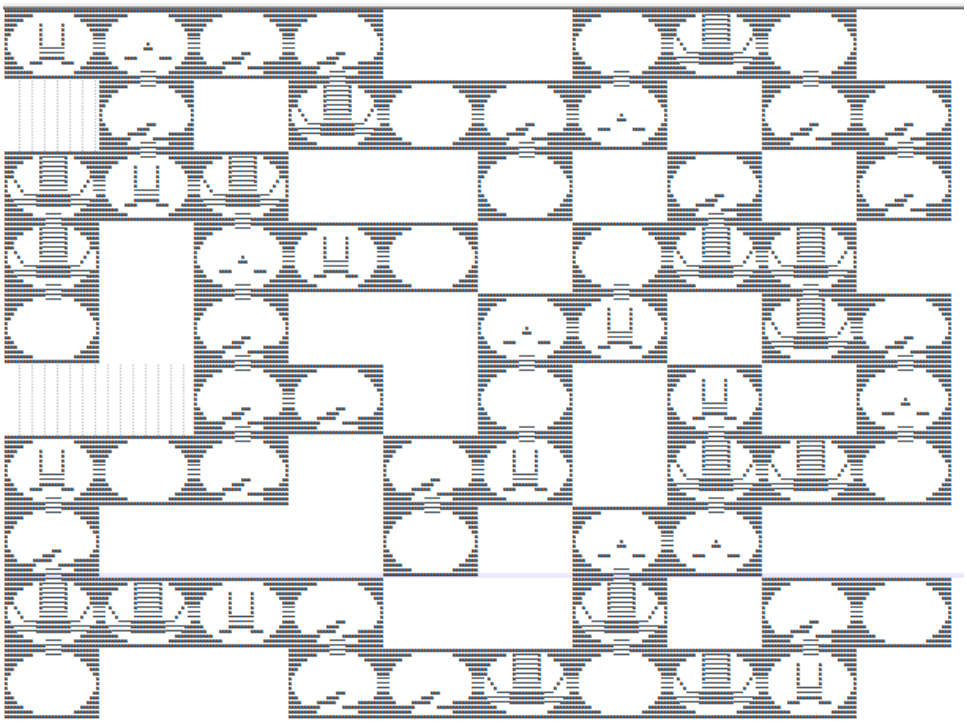


Figure 24: A fully-generated level.

As the player defeats enemies, they gain experience points. When enough experience is acquired to advance to the next level, the player gains a level-up point ([TODO: Level-up charts!!!]). Level-up points may be used to activate nodes on the level-up pentagram. A node may only be activated if an adjacent node is also activated. A special case is the first level-up, in which the player chooses one of the nodes on the inner pentagram.

Activating each node provides an increase to one or more of the player's stats. The nodes on the outer points provide +2 points to Attack, Defense, HP, MP, and Speed, while the inner nodes provide +1 points to 2 stats, determined by which outer points they are adjacent to. For example, the inner node adjacent to both the +2 HP and +2 MP nodes provides +1 to each. Although increase of stats is abstracted into a simple, consistent point system, the effects of a single point on each stat are different, and are illustrated in fig. 28.

Gaining abilities is also achieved using the pentagram. Each of the triangular points of the pentagram is associated with an ability (Fire, Dash, Lightning, Heal, Ice). Activating all nodes co surrounding that ability gives that ability to the player. The abilities are generally themed such that each one corresponds roughly to the stat associated with its triangle. For example, activating all HP nodes gives the player the Heal ability, which restores HP. Similarly, Activating

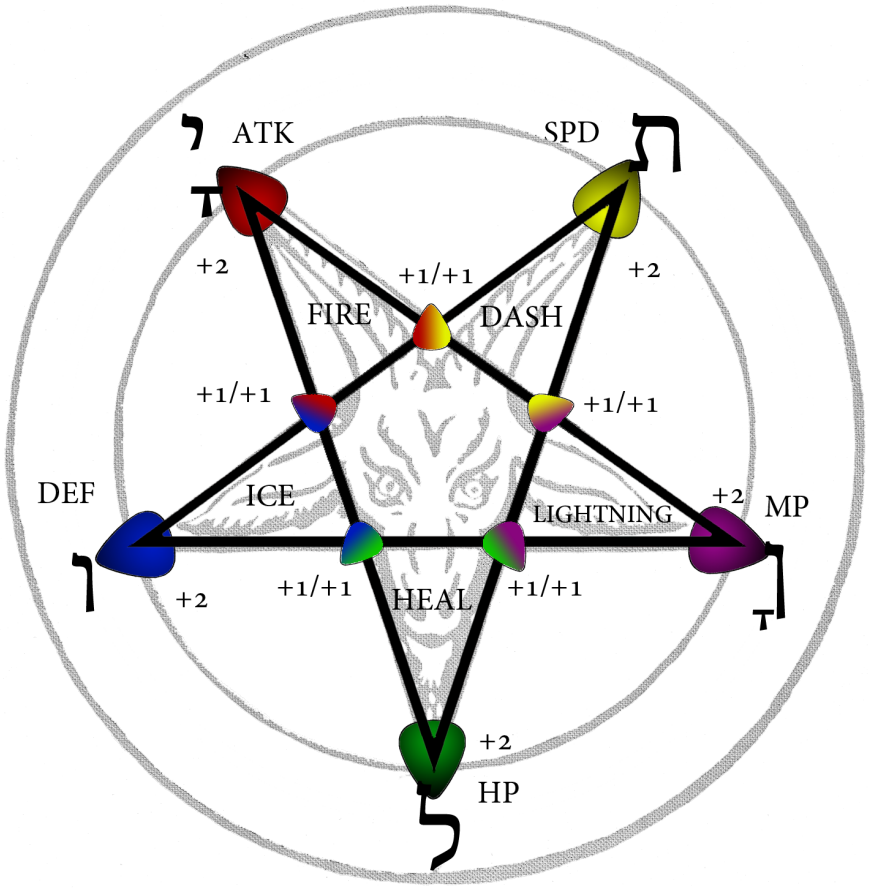


Figure 25: The level-up pentagram.

Stat	1 Point Means
HP	One extra health tank (+100 HP).
MP	One extra magic tank (+100 MP).
Speed	+10 world units per second.
Defense	+10 defense.
Attack	+10 attack.

Figure 26: Level-Up Stat Increases

all speed nodes gives the player the Dash ability, which allows fast movement.

Because the level-up system is intended to encourage replayability, a player

will not be able to activate all nodes or gain all abilities in a single play-through. Because there are a total of 10 nodes, the player will be limited to a maximum of 5 level-ups throughout the course of the game, ensuring that only a limited selection of abilities will be chosen by the player. It is worth noting that the Pentagram was initially intended to be much more extensive, with each triangle segmented into 4 sub-triangles, giving a total of 25 nodes and 20 unique abilities. However, due to limited time and resources, as well as the fact that only a demo version is being developed, the pentagram was reduced in size.

3.9 Controls

The primary intention when designing controls for this game is to keep them as simple as possible, while maintaining a degree of flexibility that allows for interesting and unique gameplay strategies. Another goal is to ensure that we use a scheme that experienced gamers are used to, allowing them to pick up gameplay easily. Figure 27 summarizes the controls.

Like first person shooters, the WASD keys are used as the primary movement keys. The A key moves the player left, and the D key moves the player right. Since the down direction has little purpose in a side-scroller, the S key is used for ducking.

Jumping is accomplished using the space key. There will be a short delay after pressing the jump key. Depending on how long the player holds the jump key, their upward velocity will be affected, easily allowing the player to control the height of their jumps. It is also fully possible to control the in-air trajectory by moving left and right. Although this is unrealistic, experience has taught us that this is the preferred way to control jumping in a side-scroller. Wall-jumping is accomplished by jumping when next to a vertical wall. This can be done indefinitely (similar to wall jumping in Mega Man X). A high-jump can be accomlplished by crouching before jumping.

Despite being a sidescroller, the inclusion of the ability to aim inspired us to use the conventions of first-person shooters. Aiming and attacking is done with the mouse. The left mouse button executes the primary attack, which may be aimed in any direction. Additionally, the orientation of the player's character (left or right). is determined by the position of the mouse. This enables the player to move forwards and backwards easily while aiming at any desired target.

Special abilities are used with the right moust button. There is at most one active special ability, which may either be selected by using the number keys or by using the mouse's scroll wheel. This scheme should be familiar to most first-person shooter players, who are used to selecting weapons in this way. The fire ability functions much like a normal attack, and can similarly be aimed in any direction. The lightning ability is the only projectile ability, and can be aimed at any target on the screen. Ice and heal cause effects local only to the player, and therefore aiming has no effect on them.

Dash is a special case of abilities. Rather than being selected and used with the right mouse button, it is mapped to movement keys. The Q and E keys

allow left- and right-dashing respectively. They were chosen due to their close proximity to A and D, the left and right movement keys.

Key	Action
A	Move left
D	Move right
S	Crouch
Q	Dash left
E	Dash right
Space	Jump
Mouse	Aim
Left Mouse	Attack
Right Mouse	Special ability
Mouse Wheel	Change special ability
1	Select Fire
2	Select Lightning
3	Select Ice
4	Select Heal

Figure 27: Summary of Controls

Stat	1 Point Means
HP	One extra health tank (+100 HP).
MP	One extra magic tank (+100 MP).
Speed	+10 world units per second.
Defense	+10 defense.
Attack	+10 attack.

Figure 28: Level-Up Stat Increases

3.10 Data Management and Flow

As this is a roguelike game which includes permanent death, there is no permanent save option. However, as challenging or frustrating as this may be to players, we do not wish to frustrate them further by interfering with their lives. To this end, a quick-save feature will be implemented, which allows a player to suspend the game through use of a "save and quit" option. When the game is restarted, the player has the option to continue from their saved game. When the continue option is selected, the saved game is deleted.

The quicksave will save the state of the world in a simple text file. The randomly-generated level structure can easily be output to and read from a string (see section 6 for details on string representation of levels). In addition to this, only a few other key pieces of information are required: enemy data and player data.

Enemy data will record the current location, stats, and state of each enemy object in the game. Position will be represented as a simple tuple of 3 floating-point numbers (i.e. (1,0,0)). The stats object will also have a simple string representation, which can be translated at run-time into a stats object. As C# provides default conversion of enumerated types to strings, storing entity states will be as simple as writing the corresponding string value.

Player data is slightly more complex than enemy data. While the simple matters of stats, state, and location are the same, the player also has experience and abilities. The save file will include a special section dedicated to these aspects. First of all, the current level and amount of experience gained is stored in the file. Second, in order to track progress in the level-up pentagram, each node of the pentagram will be given a name (i.e. HP_MP could represent HP/MP +1 / +1 node). A list of these nodes will be stored in the section, allowing the corresponding nodes to be activated upon loading the game.

At this time, there are no plans to use encryption methods to protect the game's save data.

4 User Interface

4.1 Game Shell

Game gui is rather simple and straight forward. This section includes the diagram of the main screens and a short description of each screen.

4.2 Play Screen

The play screen consists of two different parts. The first lower part is the game scene. Only a single room is shown to the player at any given time. The player can go to other rooms by digging through the snow which connects two neighbouring rooms. In each room there might be one or several enemies. Each room consists of several blocks of ice and rocks and has architecture based on one of the pre-defined room types.

Player Status The top part of the game scene is used for presenting the player information. The left corner displays HP and MP tanks (each worth 100 HP or MP units). This number of HP tanks is limited, but may increase as a level up reward. MP is also a limited resource for the player. It is important to display this so that the player understands this limitation.

In the middle top part of the game screen, there will be two other sections. The first one is the ability line which displays graphical representation of all possible abilities that the player can achieve thorough level-ups. Locked and unlocked abilities have different colors; moreover only the active one is highlighted. Player can achieve all abilities visible in figure 25. The graphic representation for all of these will be in this section.

On the right-hand side of the top of the screen, a small local map is displayed. This includes the current room and all rooms immediately adjacent to the it.

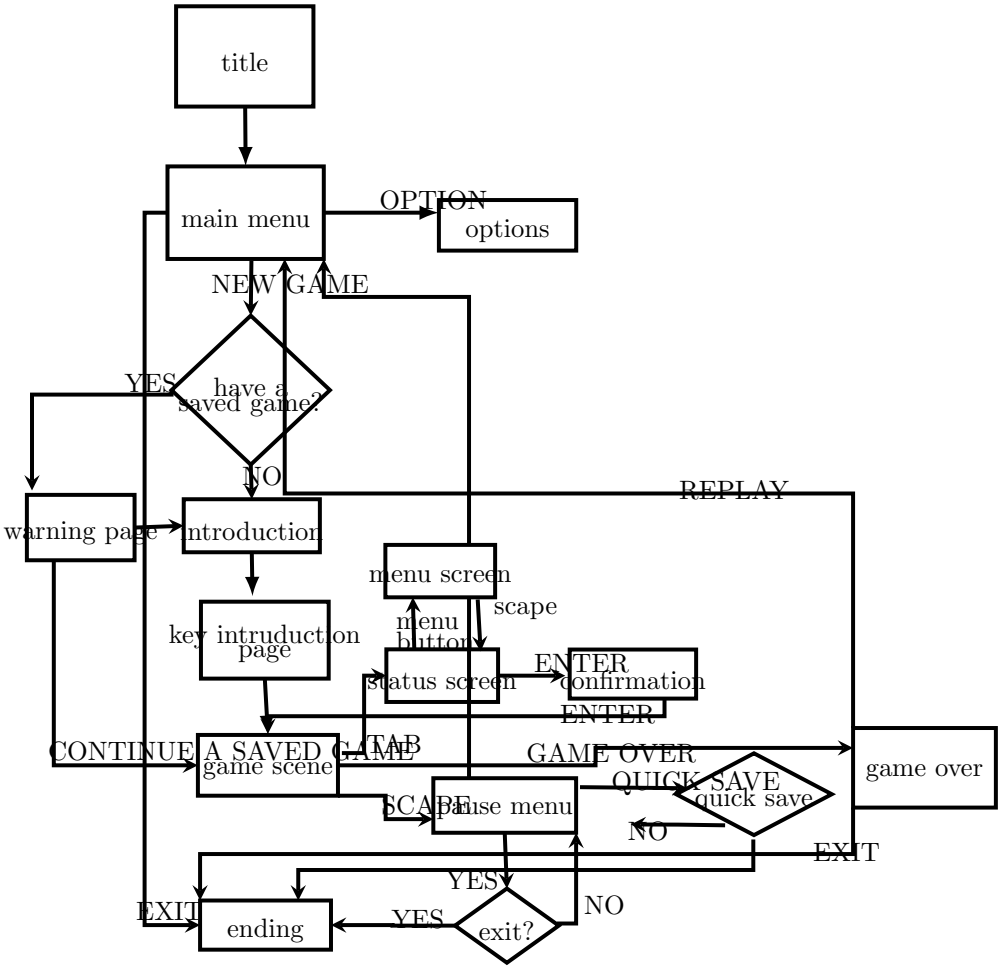


Figure 29: Game screens FSM.

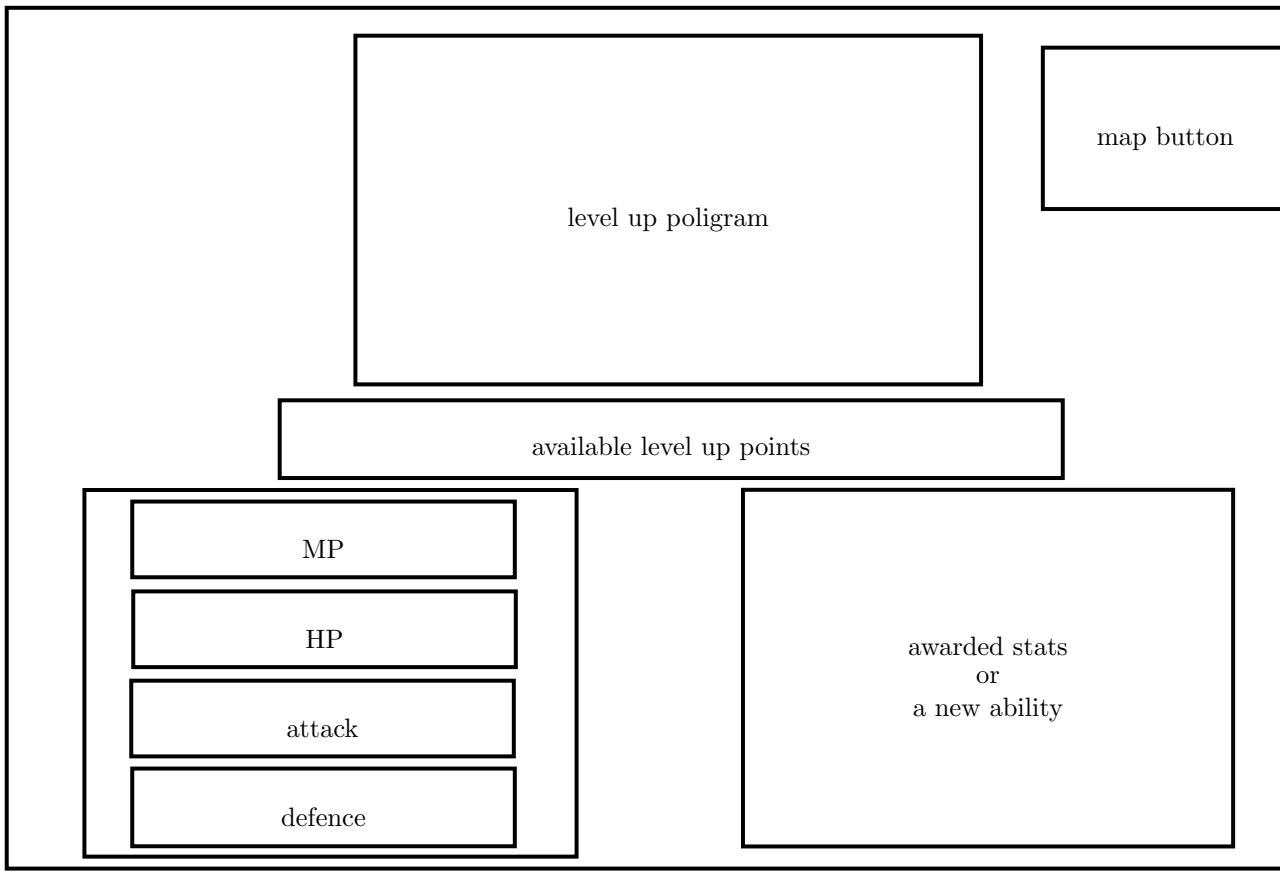


Figure 30: Level-Up Screen.

5 Technical Risk

6 Player and Enemy Statistics

Entity	HP	MP	Attack	Defense	Speed	Experience Gained
Wyrm	30	0	10	0	10	1
F. Headman	60	0	30	0	20	5
Shadowman	100	0	60	10	20	10
Snowman	200	0	60	20	0	20
Shoggoth	400	0	100	50	5	40
Xl'Grhthbtrg	1500	0	100	30	30	100
Player (Initial)	299	299	20	10	10	N/A

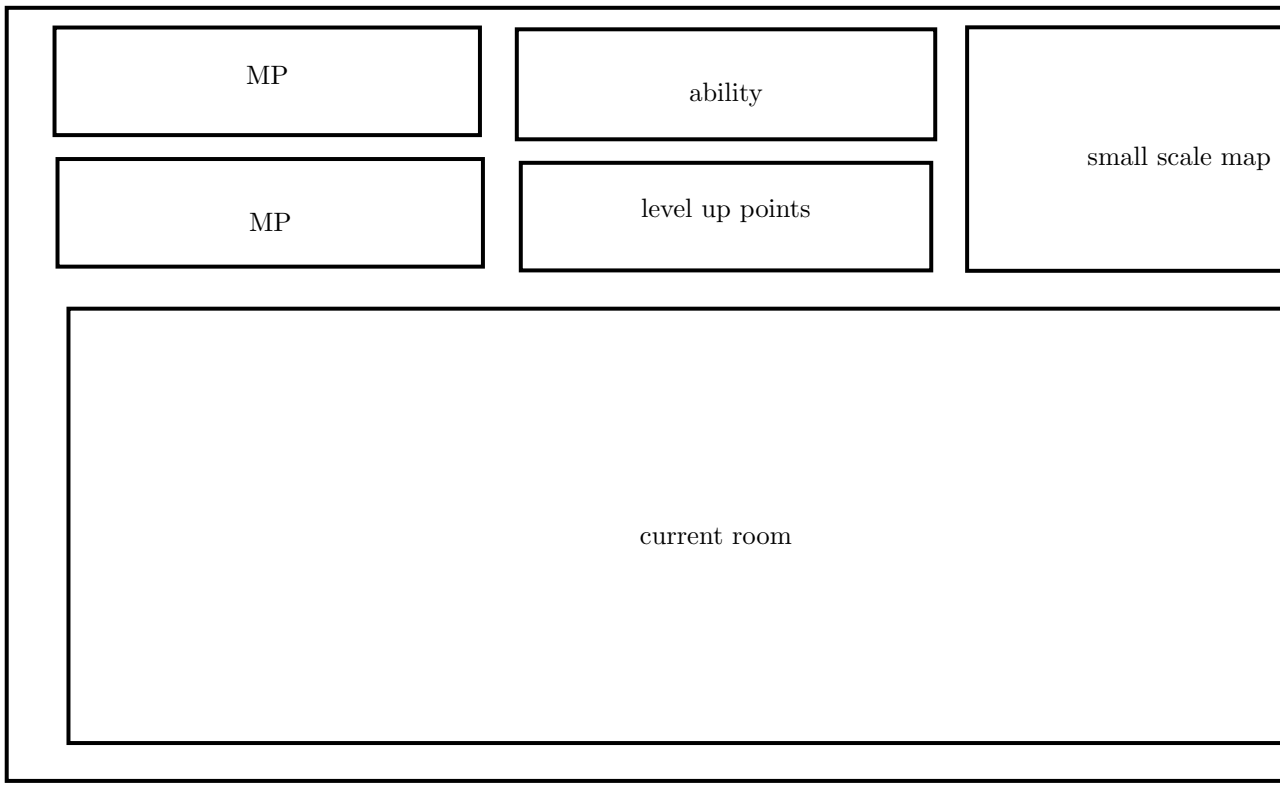


Figure 31: Game Scene.

Level-Up	Experience Required
1	50
2	100
3	200
4	400
5	800

7 Room and Enemy Configurations

Legend	
Space	Empty space
	Snow
@	Rock
w	Wyrm
f	F. Headman
s	Shadowman
n	Snowman
h	Shoggoth
X	Xl'Grhthbtrg

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@@                  @@@@@@@@@
@@@@@@                        @@@@@@
@@@@                            @@@@
@@@                              @@@
@@                                @@
@                                  @
@                  @               @
@                 @@@              @
@@                                 @@
@@@       @@@@          @@@@      @@@
@@@@                            @@@@
@@@@@@                        @@@@@@
@@@@@@@@@@                  @@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```

Figure 32: Room type 1.

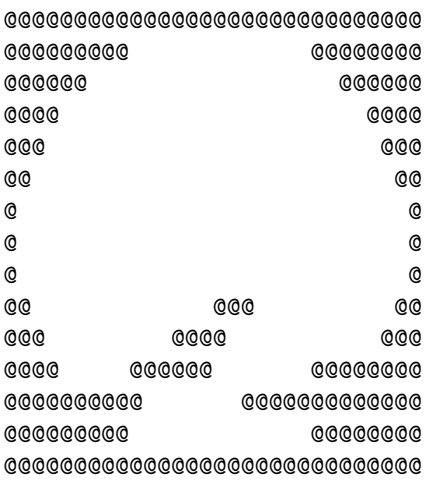


Figure 33: Room type 2.

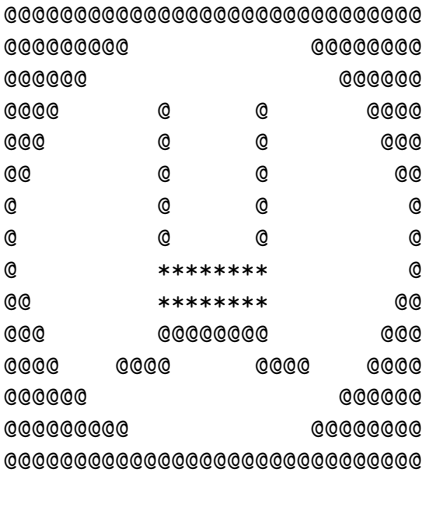


Figure 34: Room type 3.

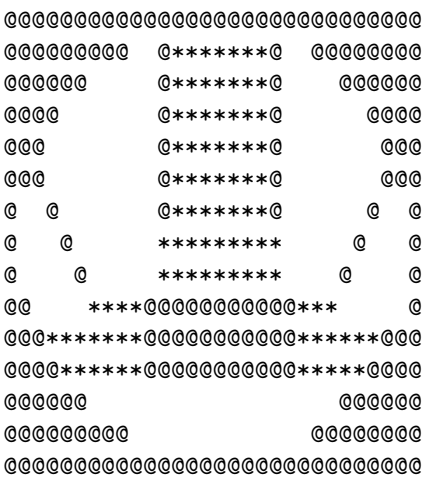


Figure 35: Room type 4.

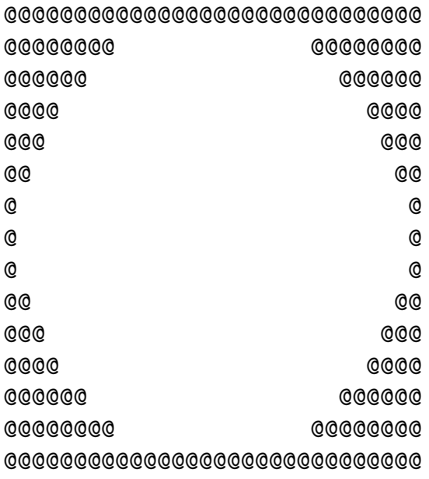
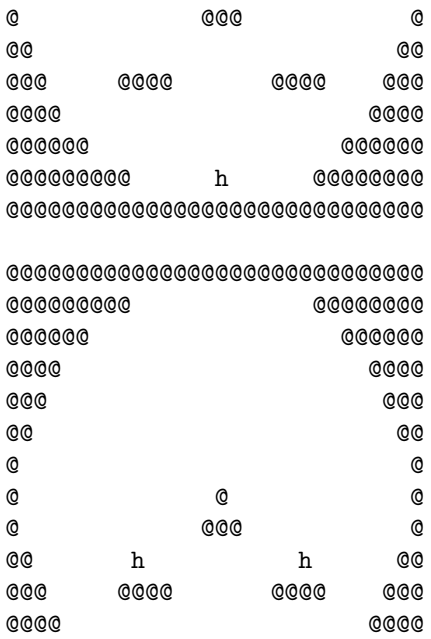


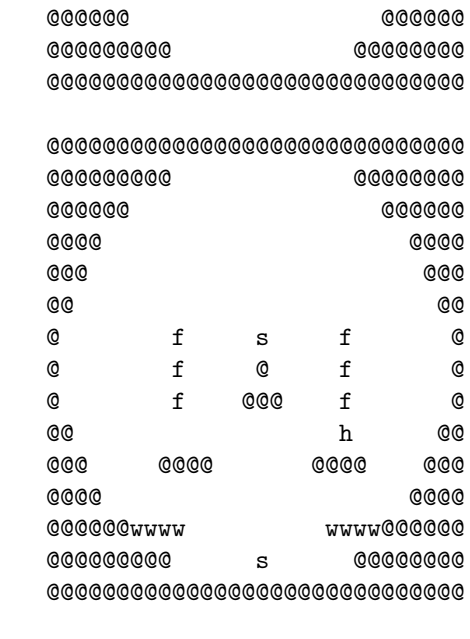
Figure 36: Room type 5.

Room 1 Configurations

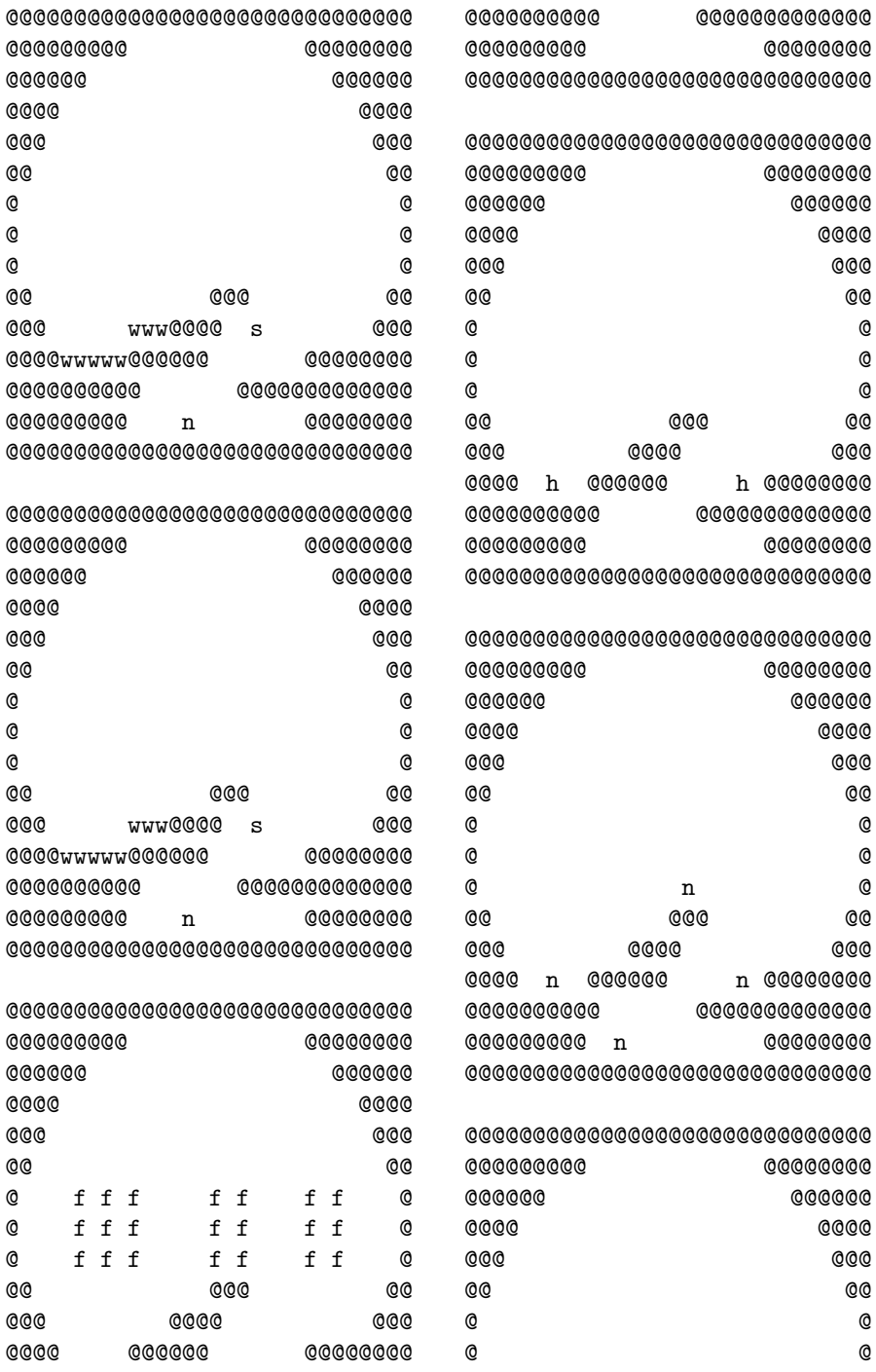
```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@                  @@@@@@
@@@@                      @@@@
@@@            f           @@@
@@      f            f      @@
@                            @
@              @             @
@       f     @@@    f       @
@@                          @@
@@@     @@@@       @@@@    @@@
@@@@           f          @@@@
@@@@@@                  @@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@                  @@@@@@
@@@@                      @@@@
@@@                        @@@
@@                          @@
@              s             @
@              @             @
@             @@@            @
@@        s         s       @@
@@@     @@@@       @@@@    @@@
@@@@                      @@@@
@@@@@@         s        @@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@                  @@@@@@
@@@@           f          @@@@
@@@                        @@@
@@                          @@
@        f           f       @
@              @             @
@             @@@            @
@@       n           n      @@
@@@     @@@@       @@@@    @@@
@@@@                      @@@@
```

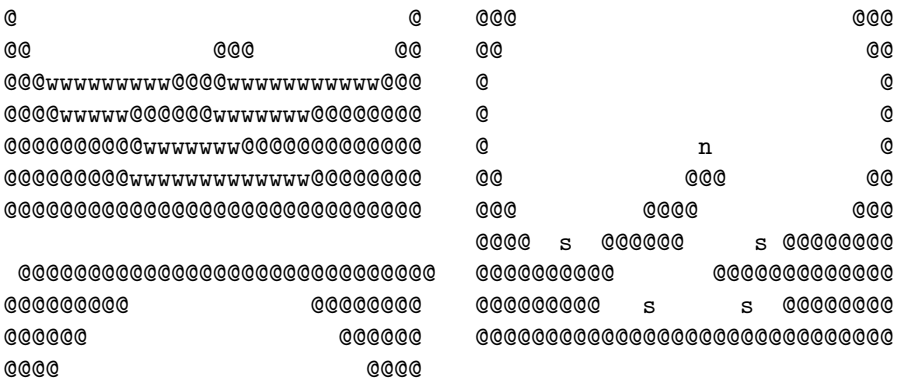
```
@@@@@@    f        f    @@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@                  @@@@@@
@@@@                      @@@@
@@@                        @@@
@@                          @@
@                            @
@              @             @
@             @@@            @
@@      wwww       wwww     @@
@@@     @@@@ wwwww @@@@    @@@
@@@@        wwwwwww       @@@@
@@@@@@wwwwwwwwwwwwwwwwww@@@@@@
@@@@@@@@@wwwwwwwwwwwww@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@                  @@@@@@
@@@@                      @@@@
@@@                        @@@
@@                          @@
@                            @
@              @             @
@             @@@            @
@@                          @@
@@@     @@@@       @@@@    @@@
@@@@                      @@@@
@@@@@@                  @@@@@@
@@@@@@@@@      h      @@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@                  @@@@@@
@@@@                      @@@@
@@@                        @@@
@@                          @@
@                            @
@              @             @
```


h
h


f
s
f
f
@
f
f
f
h
s

Room 2 Configurations

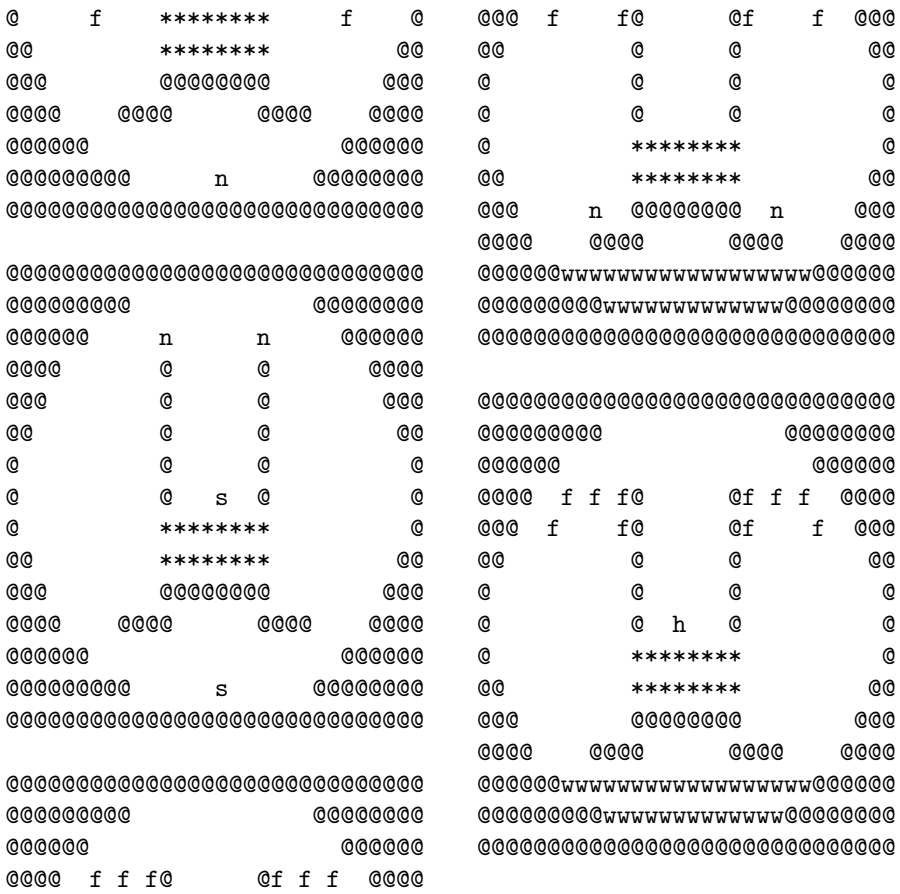




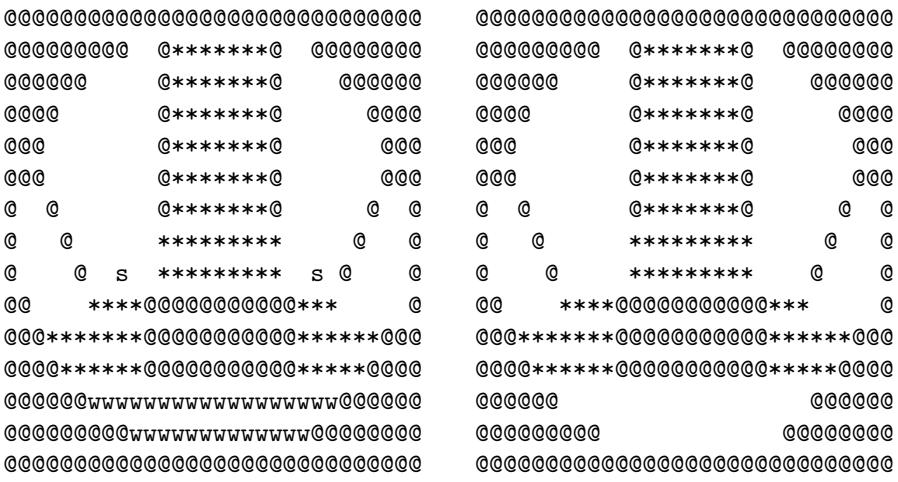
Room 3 Configurations

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@                  @@@@@@
@@@@       @      @       @@@@
@@@        @      @        @@@
@@         @      @         @@
@          @      @          @
@          @      @          @
@          ********          @
@@         ********         @@
@@@        @@@@@@@@        @@@
@@@@    @@@@      @@@@    @@@@
@@@@@@wwwwwwwwwwwwwwwwww@@@@@@
@@@@@@@@@wwwwwwwwwwwww@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@                  @@@@@@
@@@@       @      @       @@@@
@@@        @      @        @@@
@@         @      @         @@
@          @      @          @
@          @      @          @
@          ********          @
@@         ********         @@
@@@       n@@@@@@@@n       @@@
@@@@    @@@@      @@@@    @@@@
@@@@@@wwwwwwwwwwwwwwwwww@@@@@@
@@@@@@@@@wwwwwwwwwwwww@@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@                  @@@@@@
@@@@       @      @       @@@@
@@@        @      @        @@@
@@         @      @         @@
@          @      @          @
@          @  h   @          @
@          ********          @
@@         ********         @@
@@@        @@@@@@@@        @@@
@@@@    @@@@      @@@@    @@@@
```

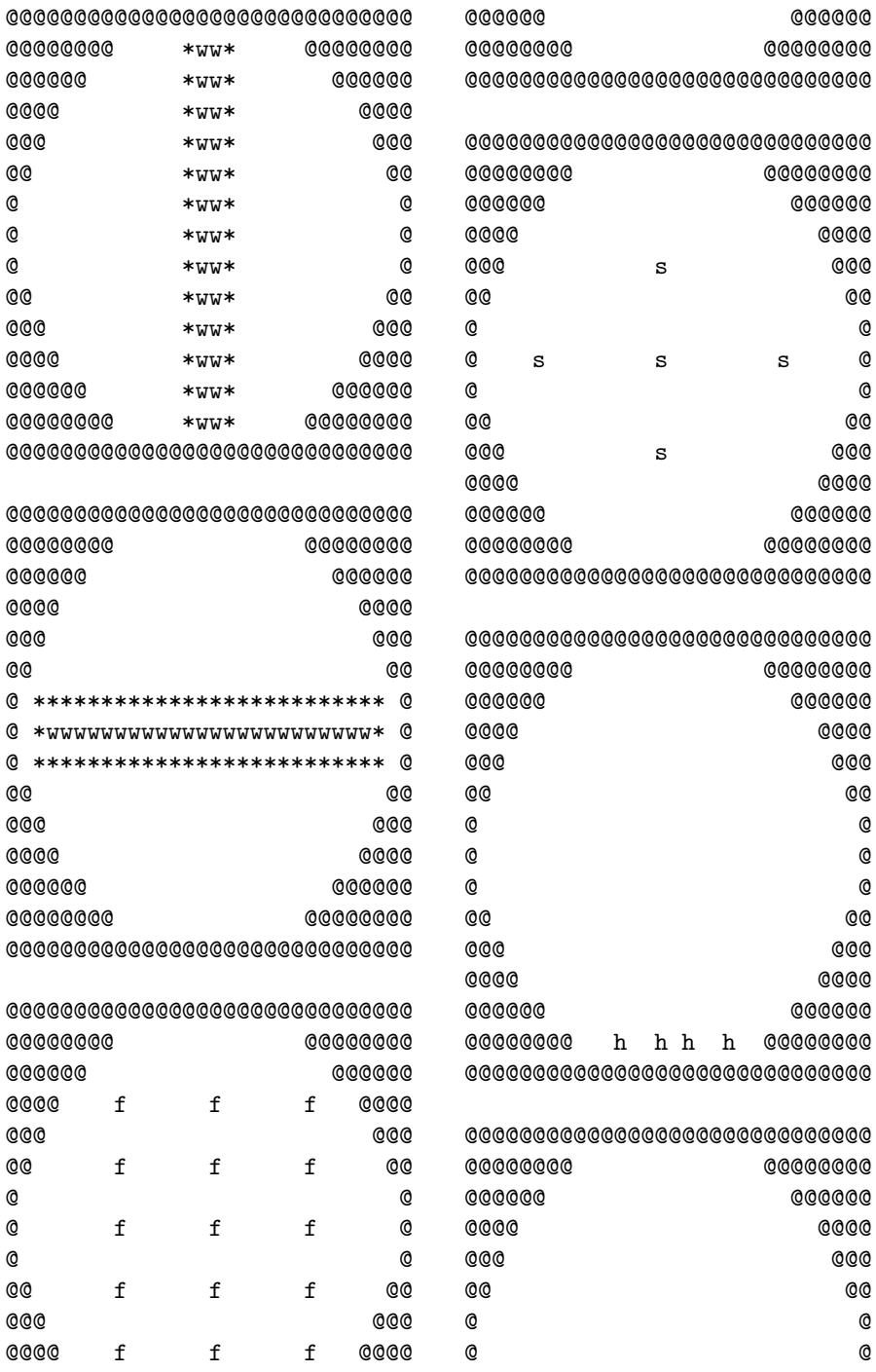
```
@@@@@@s                s@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@                  @@@@@@
@@@@       @      @       @@@@
@@@        @      @        @@@
@@         @      @         @@
@          @      @          @
@          @  h   @          @
@          ********          @
@@         ********         @@
@@@        @@@@@@@@        @@@
@@@@    @@@@      @@@@    @@@@
@@@@@@                  @@@@@@
@@@@@@@@@      h      @@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@                  @@@@@@
@@@@       @      @       @@@@
@@@        @      @        @@@
@@         @      @         @@
@          @      @          @
@          @  nn  @          @
@          ********          @
@@         ********         @@
@@@        @@@@@@@@        @@@
@@@@    @@@@      @@@@    @@@@
@@@@@@                  @@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@@@@@@@@@             @@@@@@@@
@@@@@@                  @@@@@@
@@@@       @      @       @@@@
@@@        @      @        @@@
@@         @      @         @@
@     f    @      @     f    @
@     f    @      @     f    @
```

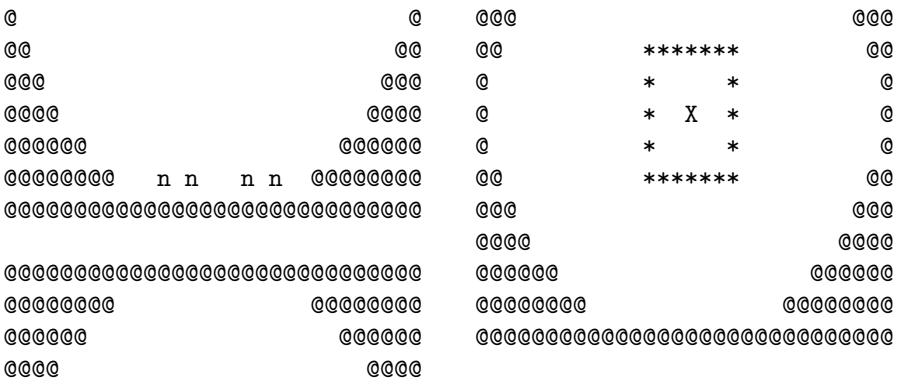

f
n
s
h

Room 4 Configurations



Room 5 Configurations




n n
n n
X