Flashlight

*Not real game footage. Example inspiration. Credits: www.moddb.com/games/forsaken-soul

Game Design Document
By Kim Kane

Course: Graphics and Imaging **Date:** 17th January 2018 **Student ID:** 15021826

Table of Contents

Introduction

Theme, Setting and Genre.....

Flashlight is a 3D first person open-world horror survival game. It is set in a dark forest, where the player is surrounded by trees, monsters and collectible items. The theme is built around sleep paralysis and the struggles and strifes of a games developer.

Brief Project Description.....

Flashlight is a 3D open-world horror survival game, explored from a first-person camera view (the player). The player has to work their way through the world (forest) and find clues in order to survive. Each clue (which will be written on a piece of paper, torn from a diary), has a puzzle the player must solve in order to find the next piece of paper. The player is equipped with a flashlight, to help them see through the heavy dark fog in the forest. They must stealth through the game dodging monsters, otherwise they will be chased by the monsters. The game ends when the player finds all the torn pieces of paper from the diary, and finds the key to the 'hidden' cabin in the woods.

Project Scope.....

I believe the game will take roughly three months to make, with a personal deadline of 1st May 2018.

Story and Gameplay

Story.....

The main story will be based around sleep paralysis – the scary feeling of being conscious, but unable to move. It is during these early stages of sleep (known as REM) that a person begins to conjure up images in their mind, believing that they are real. The player is succumb to sleep paralysis and is in fact dreaming – trying to make sense of the fear they are feeling. They have the feeling of being trapped in a permanent nightmare. This plot twist is unbeknown to the player until the end of the game.

The player “wakes up” in a dark forest, surrounded by trees and small hills. Their vision is distorted, due to the heavy fog surrounding them. The player will journey through the hilly terrain in search for a way out, trying to escape this nightmare.

A 2D Task pop-up window will appear – alerting them on what they need to do. The first task will be to find a flashlight. This will aid them in exploring the map, allowing them to see through the thick haze of fog. Upon finding the flashlight, the next task will appear – advising them to find clues to find out where they are, what is going on, etc.

It is during this time the player will begin to explore the map in-depth, looking for clues. They will come across pieces of paper torn from a diary – dotted all over the map. Each page they find has a clue on where to find the next page. The player has to find all the pages of the diary before they can progress to the next stage of the game.

However, in their hunt for the clues, they will come across many monsters – called “Nightmares”. The player must stealth their way through these monsters and not be seen. They must remember to switch off their flashlight while doing so. If the monsters see the player, they will chase the player, until they reach their 'world boundary', where they will stop. However, the monsters will be fast, and so the player will naturally want to avoid them or risk losing the game.

Upon being hit by a monster, the screen will be covered in a reddish tint to reflect the players current health. The more red the screen is, the closer they are to dying.

Upon finding all the pages of the diary, the final page will lead the player to a hidden area of the forest – where the cabin key is located. Only when the player finds this cabin key does the cabin then appear at a random location in the

forest. Once they have found the cabin they enter it safely and that is when their sleep paralysis ends and they wake up.

As well as the main plot twist of the player being some-what asleep and dreaming – there is another plot twist upon them completing the game.

They find out that the diary pages they were dreaming about belonged to them – they contained notes on a current game they were developing, trying to think of new ideas.

The game is meant to take the player through the psychological game development process, start to finish of what it is like developing a game. The nightmares are metaphoric for the obstacles the game developer faces – losing sleep over bugs or deadlines, or fear of not being successful (which can sometimes lead to sleep deprivation and then sleep paralysis) and racking their brain to come up with new, original ideas, making notes in their scrapbook or diary.

The cabin imitates the final deadline – finally completing the game.

Gameplay.....

The player will use either a PS4 controller or the mouse and keyboard on a PC to move. The current options I'd like to make available to them are: toggling the mini-map on or off, stealth mode, flashlight on/off and menu settings. The player can rotate their view and I would also like to have a 'zoom in/out' option. Their will be a mini task window that pops up with the players current task, which again can be toggled on or off. They will have an inventory that will store all of the diary pages they have collected within it, allowing them to revisit diary pages if they have forgotten the clue. There will be checkpoints throughout the game that will keep track of the players progress automatically.

Core Gameplay Mechanics.....

Movement
The player can either use a PS4 controller or a PC keyboard and mouse to move around and rotate their view. They will also be able to 'stealth' (crouch), to hide from monsters, as they will not be able to fight them.

Collecting items
The player will be able to collect items by using the mouse, by hovering over the item and then clicking on it (the mouse cursor will change its icon, to let the player know this item can be picked up). The PS4 controller will work in the same way. The items will be added to the players inventory, and then the player can view these items by accessing their inventory.

Monsters
The monsters will be controlled by means of the A* pathfinding algorithm. They will 'chase' the player, if the player gets too close and isn't in stealth mode.

Terrain
The open-world terrain will be generated using a height map, therefore allowing the terrain to be all different heights. Multi-texturing will also be implemented.

Collisions
Sloped collision will need to be implemented for the terrain. I have come across a few references that explain this in detail (see Bibliography section), however will need to research this further. Due to the game having sloped terrain, I will probably be using sphere bounding boxes for my monsters and items.

Influences.....

Looking on to the mountains in Skyrim – subtle fog and blur effects as objects get further away

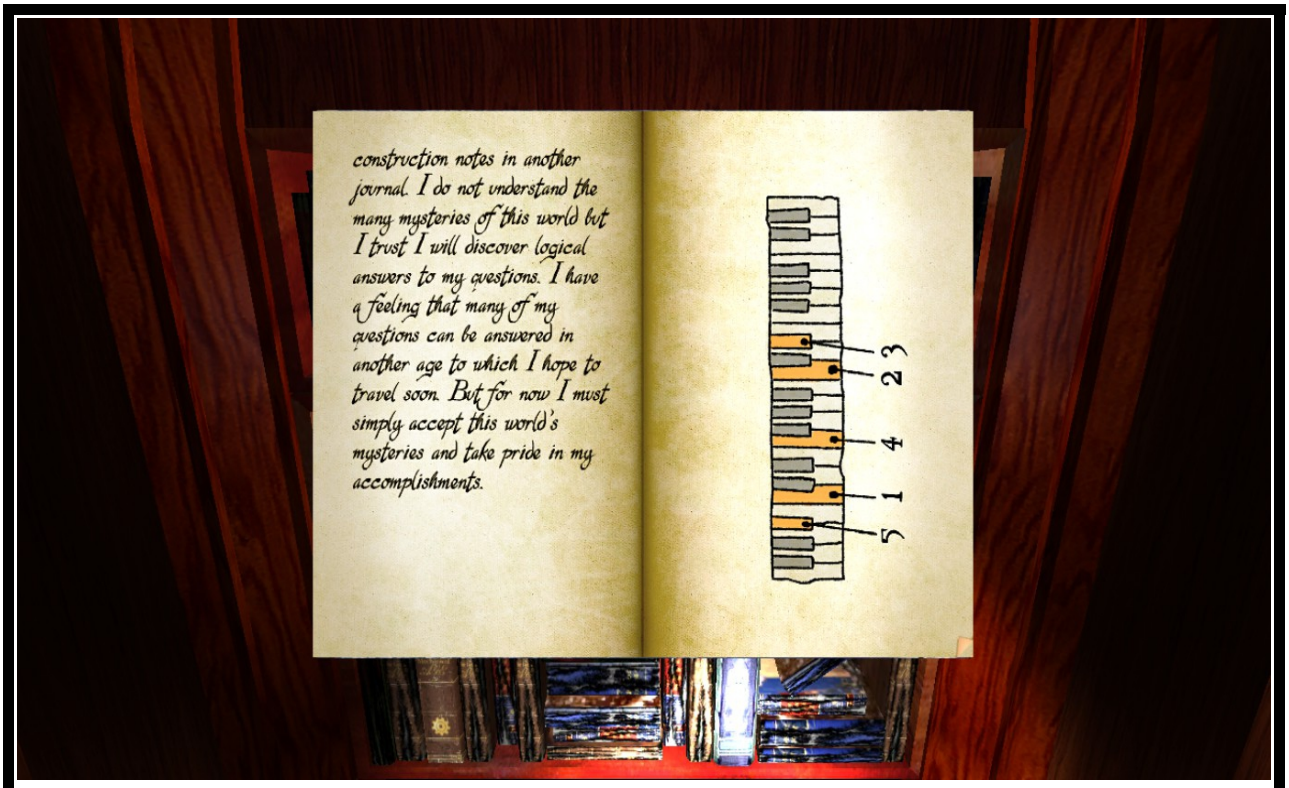
Elder Scrolls : Skyrim (3D open-world game)

Skyrim is my inspiration for the open-world terrain. The terrain has multi-texturing and is of varying levels, the most impressive of which is the mountains, which the player can climb up (it takes a long time), it generates the feeling of mystery within the game, wanting to explore every area to find what is hidden within.

The main character, Aloy, stealthing around the monsters in the distance

Horizon Zero Dawn (3D RPG)

Horizon is probably best known for their monsters being machines, that the player either hunts down or stealth's through. As there are so many monsters, each stronger than the last, it makes the player want to stealth, as opposed to fighting them. Trying to fight the monsters in the game takes patience and skill and so the player automatically feels apprehensive when they approach an area littered with strong monsters. This feeling of apprehension within the player is what I want them to experience when they run into a group of monsters in the game.



One of the many puzzles found in the game showing some piano keys that needed to be pressed in order to find another clue. However it comes with a note, which you also have to pay attention to

Myst: Masterpiece Edition (3D Puzzle game)

Myst is an extremely old game that not many have heard of. I played this game when I was younger and it was the first serious puzzle game I ever played. You are on an island, which connects to other islands, and you have to work your way through each island to find clues. These clues have a puzzle on them that you must solve to find the next clue. This is the main influence in my game – the player will have to find the clue, solve the puzzle and then find the next clue. The final clue leads them to the end of the game.

A dark corridor in Outlast, showing eerie lighting effects

Outlast (3D escape game)

Outlast is a horror game, set in an old run-down asylum that the player must escape. Again, the player must find clues (blue documents) dotted around the asylum, dodging the monsters in the process. The Player cannot fight back and has nothing but a camcorder on them. This game influenced me because of the FP camera they use, and the lighting effects used - projecting a dark ambient light over every corridor to give the game a scary vibe. The camcorder is what inspired me to equip the player with with a flashlight, hence the name of my game.

2D Assets Needed.....

Textures

- Tiled Environment Textures – grass, rocks, dirt
- Menu backgrounds
- Buttons

Heightmap data

- A 2D grayscale image of a heightmap, custom made or downloaded

Inventory

- A HUD inventory and task window pop-up

Skybox

- A 2D cube map image to represent the sky

Cursors

- Unique cursor icons used when collecting items

3D Assets Needed.....

NPC

- Monster – "Nightmares" – A 3D animated model of a shadow-like monster

Environment

- A textured cabin model
- Trees
- Transparent Fern/grass

Sound.....

Theme tune

- Milla Jovovich – Flashlight
- Pitch Perfect 3 – Flashlight

Character movement

- Sound to imitate someone walking on grass

Flashlight

- 'Click' sound when turning on and off

Monster

- Sounds when attacking and stationary

HUD

- Sounds when entering/exiting the 2D HUD, clicking buttons, closing the mini-map or task popup windows

Ambient sound (in-game)

- Wind, distant sound of eerie noises

Shaders.....

Flashlight

The flashlight will be attached to the camera (player) and follow the cameras position and direction. Below is some sample code of how the lighting will be organised in a shader file:

```
struct BaseLight
{
	vec3  color;
	float intensity;
};

struct PointLight
{
	BaseLight   baseLight;
	Attenuation attenuation;
	vec3        position;
	float       range;
};

struct SpotLight
{
	PointLight pointLight;
	vec3 direction;
	float margin;
};
```

The PointLight will derive from a BaseLight, and the SpotLight will derive from the PointLight, within the engine code. A spotlight object will then be created and its transform position in the 3D scene will mimic that of the camera/player.

I have not yet built shaders to handle fog, terrain, multi-textures or gaussian blur. However, these will be further shaders I will use.

Code.....

Character Scripts (Camera Movement/Input)

```
void Camera::Move(Vector3f direction, float amount)
{
	m_position = m_position.Add(direction.Multiply(amount));
}

void Camera::RotateHorizontal(float angle)
{
	Vector3f horizontalAxis = s_defaultVerticalAxis.Cross(m_forward).Normalized();
	m_forward = m_forward.Rotate(horizontalAxis, angle).Normalized();
	m_up = m_forward.Cross(horizontalAxis).Normalized();
}

void Camera::RotateVertical(float angle)
{
	Vector3f horizontalAxis = s_defaultVerticalAxis.Cross(m_forward).Normalized();
	m_forward = m_forward.Rotate(s_defaultVerticalAxis, angle).Normalized();
	m_up = m_forward.Cross(horizontalAxis).Normalized();
}
```

Above is the main functions used to move and rotate the camera/player. I am using directional vectors for the camera/player movement. At present, I am doing all transformations of the camera/player using euler angles. Due to the problems with gimble lock, I will later use quaternions for my camera/player movement. Some examples of how these functions can be used for keyboard, mouse and PS4 controls are below:

Rotating the camera left using the left mouse button

```
if (Input::Instance()->MouseButtonPressed(SDL_BUTTON_LEFT))
{
	m_camera->RotateVertical(-InputConstants::RotateSpeed);
}
```

Moving the camera left using the left controller axis

```
m_camera->Move(m_camera->GetLeft(), -Input::Instance()->GetControllerLeftAxis().x);
```

Moving the camera left using the left arrow key on the keyboard

```
if (m_keys[SDL_SCANCODE_LEFT])
{
	m_camera->Move(m_camera->GetLeft(), InputConstants::Speed);
}
```

Transform Hierarchy & Game Objects Scripts

```
GameObject* GameObject::AddChild(GameObject* child)
{
	m_children.push_back(child);
	child->m_localTransform.SetParent(&m_localTransform);
	return this;
}
```

The AddChild() function is the most important function in the transform hierarchy. It allows us to attach objects to other objects, in its simplest explanation, and means we can transform around those objects, instead of in local 3D space. An example of how the function is used is below:

```
rootObjectExample = new GameObject(Vector3f(-1.5f, -2.0f, 10.0f), meshFontExample);

rootObjectExample->AddChild(gameObjectLaraExample = new GameObject(Vector3f(-10.0f,
-1.0f, 0.0f), meshLaraExample));
```

We first create a root game object, which acts as an origin in our 3D world and will usually be at position 0, 0, 0. Then we add all game objects to this root object by means of the AddChild() function. Effectively, we now know who the parent object of the child is, and this root parent object will be responsible for updating and drawing all its children. But, we can go further than that:

```
gameObjectLaraExample->AddChild(gameObjectCubeExample = new GameObject(Vector3f(-5.0f,
-0.4f, 0.0f), meshCubeExample));
```

We can add children objects to other children objects. This child object will now be the parent of the other child object. In doing transformations this way, we can now do the following:

```
gameObjectLaraExample->GetTransform()->SetLocalPosition(Vector3f(-10.0, 1.0, move),
true);
gameObjectCubeExample->GetTransform()->SetLocalPosition(Vector3f(-5.0f, -0.4f, 0.0f),
true);
```

We set the local position of the object, and decide whether or not we want to position this relative to the parent position by passing in 'true' or 'false'. So, rather than automatically following the parents transformations, we now have the ability to toggle this on or off whenever we want. True = follow parent, False = transform around your own axis in the 3D world.

How this function looks within the Transform class is as follows:

```
void Transform::SetLocalPosition(const Vector3f& position, bool setRelativeToParent)
{
	m_positionSetRelative = setRelativeToParent;
	m_position = position;
	if (setRelativeToParent) { SetRelativeToParentPosition(); }
}
```

If we want to set the local position of the object relative to the parent, then we call the function SetRelativeToParentPosition(), otherwise, the position of the object will be the position we pass in to the function and the object will position itself relative to the origin only – 0, 0, 0 (the center of our world).

```
void Transform::SetRelativeToParentPosition()   const
{
	GetParentMatrix().Transform(m_position);
}
```

The SetRelativeToParentPosition() function is rather simple – it takes the parent matrix and basically multiplies it (in simplistic terms) by the position vector, by use of the Transform() method.

The last thing we can do, is choose to change the parent any time we want to, by use of the SetParent() function:

```
void Transform::SetParent(Transform* parent) { m_parent = parent; }
```

There is much more happening in the Transform hierarchy, such as optimization, dirty flags and recursive functions, however this would take around 10 pages to explain and full explanations and comments can be found within the source code.

Why this is useful in the game, will be when I want to make the monsters follow the player for a period of time. I can now do this in very few lines of code, passing in a flag. Some pseudocode of how it can eventually work:

```
if(PlayerNearMonster)
{
	monster->SetParent(player);
	monster->FollowParent(true);
}
else { monster->SetLocalPosition(); }
```

In terms of the game objects, as I mentioned above, the root object is responsible for updating and drawing all of the game objects. The game objects are stored in a vector and I iterate over them every draw and update call.

```
bool GameObject::Draw(Camera* camera)
{
	m_shader.Bind();
	m_shader.UpdateAllUniforms(m_localTransform, camera);
	m_mesh.Render();

for (std::vector<GameObject*>::iterator i = m_children.begin(); i != m_children.end(); ++i)
	{

		(*i)->Draw(camera);
	}
	return true;
}
```

And upon deletion of the root object, all other objects get deleted too.

The other scripts are available in the source code. The scripts are more engine-based rather than game-specific and so I haven't added them here. I felt the above functions were the most important to show in what I am trying to achieve and the tools that will enable me to do so.

Some of the other scripts I have written are: Terrain, Mesh and Texture classes, Manager classes, ObjLoader class for loading in the models, GameState hierarchy, Shader and Buffer classes, and finally the maths classes.

Scripts I will be writing going forward are: animation, proper terrain and heightmap generation, monster movement (A* pathfinding algorithm, the monster will chase the player, but I still don't want it walking through trees!), multiple shaders, possibly a rendering engine, a binary file reader and writer for the obj files, possibly multi-threading for cut-scenes (if I can find some royalty-free ones), an Inventory class, collectible objects and a 2D Interface class. This isn't an extensive list, but these are the most important scripts I will need to write for the time-being.

Bibliography.....

I have learnt a lot in doing this project, primarily 3D maths and lighting formulas. Writing the maths classes was the best thing I have ever done in terms of programming, as it really made me understand everything so well. However, I would not have been able to do it without the following references I found in my research:

Lighting & Shaders - YouTube

ThinMatrix - 3D Java Game Development Videos (incl. Terrain and height map)
TheBennyBox - 3D Game Engine Development, Physics Engine development, OpenGL Graphics Development

Websites

www.lighthouse3d.com/tutorials/glsl-12-tutorial/point-light-per-pixel
www.tomdalling.com/blog/modern-opengl/06-diffuse-point-lighting
www.mbsoftworks.sk/index.php?page=tutorials&series=1&tutorial=16
http://pyopengl.sourceforge.net/context/tutorials/shader_5.html
www.lighthouse3d.com/tutorials/glsl-tutorial/directional-lights-per-pixel
www.lighthouse3d.com/tutorials/glsl-tutorial/directional-lights-per-vertex-ii
JOEY!! a.k.a learnopengl.com

OBJ Loading

Github example codes - A number of these I looked at for reference and took ideas from the ones I thought were the cleanest and simplest
Book - OpenGL Insights, by Patrick Cozzi and Cristophe Riccio

C++, OpenGL - YouTube

TheChernoProject - Macros, OpenGL buffers, handling OpenGL errors, C++ const correctness, 2D Game Engine Development, Sparky Engine
MakingGamesWithBen - C++, strings (for file loading)
Jamie King - OpenGL, camera, transformations, buffers, lighting effects, 3D maths
TheNewBoston - C++ tutorials

Other References (maths, optimization, etc.) - YouTube

TheNewBoston - Geometry, algebra and physics tutorials
TheBennyBox - 3D maths & transform hierarchy/scene graph, optimization
TheChernoProject - 3D maths (Sparky Engine)

Miscellaneous

Github
Stack overflow
GLM library

Summary.....

To sum up, I hope you are as excited as I am about seeing the game in its final stages. I have learnt a vast amount of things in such a short period of time and I am looking forward to putting this knowledge in to making something, hopefully, really amazing. Supporting documents, such as: a UML diagram, my personal Bug Report and Development Log, can be found in the same file as this GDD. Thanks for reading!