|  |
| --- |
| AC31009 – Games Programming |
| Shadows – Game Plan |
| Johnathan Law - 100008284 |

Table of Contents

[One Page Game Design Overview 2](#_Toc370162611)

[Detailed Game Design 3](#_Toc370162612)

[Why Shadows? 3](#_Toc370162613)

[Hardware and Software 3](#_Toc370162614)

[Implementation 3](#_Toc370162615)

[Menus 4](#_Toc370162616)

[Gameplay 4](#_Toc370162617)

[Search State 4](#_Toc370162618)

[Hunt State 4](#_Toc370162619)

[Alert State 4](#_Toc370162620)

[Graphics and Audio 5](#_Toc370162621)

[Game Graphics 5](#_Toc370162622)

[Game Audio 5](#_Toc370162623)

[Future Improvements 5](#_Toc370162624)

[Interface Wireframes 6](#_Toc370162625)

[Initial Menu 6](#_Toc370162626)

[Game Interface 6](#_Toc370162627)

[Pause Menu 6](#_Toc370162628)

[Time Management Plan 7](#_Toc370162629)

[UML Diagrams 8](#_Toc370162630)

[Activity Diagrams 8](#_Toc370162631)

[Main Menu Activity 8](#_Toc370162632)

[State Transition Diagrams 9](#_Toc370162633)

[Handling Windows 8 UI Snapping 9](#_Toc370162634)

[Player Detection 9](#_Toc370162635)

[Class Diagram 10](#_Toc370162636)

[Diagram Justification 11](#_Toc370162637)

[Behaviour Diagrams 11](#_Toc370162638)

[State Diagrams 11](#_Toc370162639)

[Class Diagram 11](#_Toc370162640)

[Technical Implementation 12](#_Toc370162641)

[Design Patterns 12](#_Toc370162642)

[Model View Controller (MVC) 12](#_Toc370162643)

[Factory 12](#_Toc370162644)

[State 12](#_Toc370162645)

[Singleton 12](#_Toc370162646)

[Game Features 13](#_Toc370162647)

[Collision Detection 13](#_Toc370162648)

[Artificial Intelligence 13](#_Toc370162649)

[Other Features 13](#_Toc370162650)

[Play Testing 13](#_Toc370162651)

[Bibliography 14](#_Toc370162652)

[Appendices 15](#_Toc370162653)

[Appendix One 15](#_Toc370162654)

[Appendix Two 15](#_Toc370162655)

[Appendix Three 15](#_Toc370162656)

[Appendix Four 15](#_Toc370162657)

# One Page Game Design Overview

##### Title

Shadows

##### Genre

Top-Down Stealth

##### Target Audience

Users looking to play competitively or enjoy stealth based games. Aimed at ages 12 and above.

##### Platform(s)

Windows 8 and Windows RT. Possible implementation on other platforms possible due to use of **MonoGame**[[1]](#footnote-1).

##### The Big Idea/Concept

A top-down stealth game where you see how long you can hold out against ever increasing enemies. Compete against your friend’s to see who can last the longest.

##### Unique Selling Points

Stealth is not commonly found in a top-down style. The victory condition and competitive play style leads to a more engaging experience.

##### Play Mechanic

In Shadows you play until you are caught by the guards, your objective is simple: stay alive. Guards move continually, so there’s no time to rest. You must actively avoid the guards either by trying to hide or keep moving, but the guards will become increasingly determined to find you.

##### Game Summary

You are Jacques, a French spy working for the ICN (International Crime Network). You’re currently on mission and someone has tipped off the enemy that you’re sneaking around their secret lair. It’s only a matter of time until you’re caught and “dealt with”, how long can you hold out?

##### Similar Competitive Products

There are some similar games out there in terms of style such as Klei Entertainment’s **Incognita[[2]](#footnote-2)** and Tom Francis’s **Gunpoint[[3]](#footnote-3)** however both of these games are more strategy based. Shadows could be more closely compared to **Pac-Man** however the victory condition in Shadows is simply to survive.

# Detailed Game Design

## Why Shadows?

Shadows is a very versatile and easy game to pick up and play. The objective is simple to understand, yet tricky to achieve. Due to the exponential difficulty (as more enemies spawn with time), players will return to play and attempt to beat their previous times. As a keen lover of stealth games, a top down stealth game is very interesting concept. By providing a challenging game and interesting art style, Shadows will hopefully prove popular with its target audience.

The game shall prove a challenge to even the most veteran players with the possibility to start with however many enemies you want, and reducing the number of crates using the map seed functionality. The game can be seen as a top down ***Splinter Cell*** where you have been detected, and now you must survive.

## Hardware and Software

Shadows shall run on any Windows 8 hardware platform (including Windows RT) as it will be created as a full screen application. The game shall allow the player to either play using the keyboard and mouse, or the Xbox 360 Gamepad. By allowing control of the game by both keyboard/mouse and the Xbox gamepad, we ensure that more play styles are covered and that if we decide to port the game to the Xbox 360 later, there will be less code to be edited.

Shadows will be programmed in C# using **MonoGame** as it allows for porting to multiple platforms and allows creation of Windows 8 full screen applications. It uses an open source implementation of DirectX called **SharpDX[[4]](#footnote-4)** in order to improve portability. I had also considered using C++ and SDL or Java and Swing, however I chose this setup in order to facilitate the distribution of the game through the Windows Store and allowed me to progress my C# knowledge.

I have chosen MonoGame over XNA as it is much more relevant and up to date, as Microsoft discontinued the XNA framework in 2011. MonoGame is an open source alternative, which provides the same functionality but is still in active development. The MonoGame framework is also much more portable as it provides support for several platforms[[5]](#footnote-5) meaning that the Shadows came could be ported to one of these platforms much easier if desired.

The Windows Store was chosen as my release platform of choice due to the ability for students to release on the Store for free using DreamSpark. I plan on releasing my game to market after completion, and the Windows operating system has the highest penetration rate meaning a much larger potential consumer base.

## Implementation

Similar to the implementation of Pac-Man, the Shadows playing area will be split into a grid. This will be used to place the obstacles (crates, buildings, etc.) and allow the AI to be implemented somewhat easier when the game falls into the **hunt** or **alert** state. Using the Factory and Decorator design patterns, allows for the possibility of additional enemy types in the future.

I shall be using Visual Studio 2012 for the majority of the development, as it provides a familiar IDE and incredibly useful debugging tools including IntelliTrace. It is very easy to write native C# in Visual Studio as it provides syntax highlighting and automatic code completion. As I have used Visual Studio 2012 for C++ and C# development in the past, there will be no hurdles in getting used to the system. This means that there will be no time required to learn the ins and outs of the IDE.

Occasionally I may have to use Visual Studio 2010 to get access to the XNA Content Pipeline which is required for the creation of the XNB files used by MonoGame. Unfortunately the creators of MonoGame have not yet found a solution to this problem, and for the time being this is the easiest way to create these files.

## Menus

After the game loads (past the initial Windows 8 application splash screen), the player shall be presented with a simple menu allowing them to launch the game with a random seed, set their own seed and launch the game and change the options. When paused, the player will be offered a similar menu allowing the options to be changed and the player a chance to quit and return to the main menu.

This simple menu system provides an unobtrusive way of starting the game and allowing the player to change the settings to suit their needs. This is the same in the pause menus, where the screen will fade out/in to/from the menu.

## Gameplay

Once the player has loaded the game, they are placed randomly into the map along with three enemies. The number of enemies increases over time, with the potential for more difficult enemies in the future. The player must move around to avoid detection. The level will comprise of a bigger area than is seen on screen, thus the camera is fixed centrally on the player. This means that there is always the danger that an enemy will be coming towards you off camera. It is planned that enemies you are not facing will be somewhat transparent as you are not looking in their direction (and clearly you do not have eyes in the back of your head).

Shadows has a number of key states for enemies, based on previous interactions with the player. They are:

* Search (The initial, and default state)
* Hunt
* Alert

Although seemingly similar, each state modifies the behaviour of the enemies. From increasing their movement speed to extending their vision distance. The rule set (how the enemy moves towards the player) is also adjusted depending upon the state of the game.

### Search State

The search state is used when the enemies on screen are unaware of the player’s location. In this mode they will randomly search the map looking for the player, with no real logic in their search pattern. When an enemy detects the player in this state, the game will alter the state to **hunt.** The player will have a cooling off period (of approximately 5-10 seconds) to escape the enemies before being captured. The enemies shall move at base speed in this state, and will not actively hunt the player, rather they shall search based on where the player has been previously (a la tracking footprints in the snow).

### Hunt State

In the hunt stage, they will have become aware of the player and begin to flock (and search around) the player’s last known location. In this stage, the player can be captured with no cooling off period. If the player falls into an enemy’s line of sight, the player will be caught and the game will end. If the player is not detected, the enemies wall fall into the **alert** state. In this state, the enemy’s movement speed shall be 2x the base speed.

### Alert State

In the alert state, the enemies are far more aggressive in terms of finding the player. Their vision cones increase in size and their speed increases 1.5 times their base speed. This phase/state lasts for 30 seconds, after which the enemies will return to the **Search** state.

It is proposed that the player may have some way of incapacitating enemies either by knocking them out from behind, or stunning using flash grenades. However, this is purely a desired feature and in no way a required ability in the game. This would provide a means of evading capture even when it seems there may be no escape.

## Graphics and Audio

### Game Graphics

Shadows makes use of sprite images in the XNB (XNA’s proprietary binary file type) format. Sprite images were chosen due to the lack of SVG support in both XNA and MonoGame thus the use of vector images would be very difficult as a SVG renderer would need to be created or sourced.

In the game, players can make use of their surroundings to hide from enemies. In the beginning, crates will be used as a placeholder for further possible images (i.e. buildings), a sample of the potential crate images can be seen in **Appendix One**. Both of these crate graphics were sourced from **OpenGameArt**[[6]](#footnote-6). As the game is being created as a Windows 8 full screen application for distribution in the Windows Store, it is important to have application icons that show the user what they are going to select in their start menu. The font **Shortcut** by **Misprinted Type**[[7]](#footnote-7) was selected for its dark, grunge look and is used for both the icons and title images. It is also free for commercial use (provided it is not directly redistributed) so we therefore use pre-made graphics that were created using the font. A sample of these graphics can be seen in **Appendix Two**.

A friend of mine who is a 4th year graphic design student at DJCAD, **Lauren Howat**[[8]](#footnote-8), has been commissioned to create improved game icons. A sample of the proposed imagery can be seen in **Appendix Three**. This look has been chosen as it is seen as a very typical spy look (trench coat and hat). All of the resources chosen have licenses allow for commercial use, providing proper attribution is made. The crate images are in the public domain, and as such require no attribution and can be used freely. It was decided to attempt to source the graphics from other sources as I am lacking in artistic talent and allows for more time to be spent for the actual programming of the game.

In order to follow the **Windows 8 design principles**[[9]](#footnote-9), we use the Segoe UI font as a Sprite Font whenever we wish to directly write to the screen (for example, instructions and buttons). The background on the main menu is very simple, a repeating square pattern was chosen and can be seen in **Appendix Four**. Overall, the graphical styling of Shadows is very dark and gritty as you are a spy who is forever hiding in the shadows. It is important to follow the design principles as it aids in usability by using the same cues that are already in place by the system.

### Game Audio

To fit in with the game style, the musical styling of the game is very ambient. For Shadows, this typically means sounds are created using synthesizers to produce a short, looped piece of audio. Alert noises will be played when you are detected by the enemy, and your movement will play footstep noises (depending upon the material being walked on).

Game audio has been mainly sourced from **FreeSound[[10]](#footnote-10)**, otherwise sounds have been created by myself for use in the project. The sounds have been chosen in order to create a tense atmosphere yet provide clear audible cues for important actions within the game.

## Future Improvements

Ideally, Shadows will use randomly generated maps based upon seeds (like **Minecraft[[11]](#footnote-11)** map generation). Improved enemy AI with additional enemy types would be a notable improvement as it would add to the re-playability of the game. A power-up feature is also a possibility. After the player survives for a certain amount of time, they would be awarded power ups. For example, the ability to see the enemy’s vision cones or to become shielded.

A 2nd player could also be added, where you would compete competitively to see who would get caught first (rather than the intended asynchronous “multiplayer” that is planned by the sharing of map seeds). As the game runs on Windows RT, another improvement would be to allow control using the touchscreen. This would require a method to control the game on the screen, such as virtual controller.

# Interface Wireframes

## Initial Menu

SHADOWS

Play Random Game  
Play Game with Seed

Game Options  
Exit Game

## Game Interface









## Pause Menu

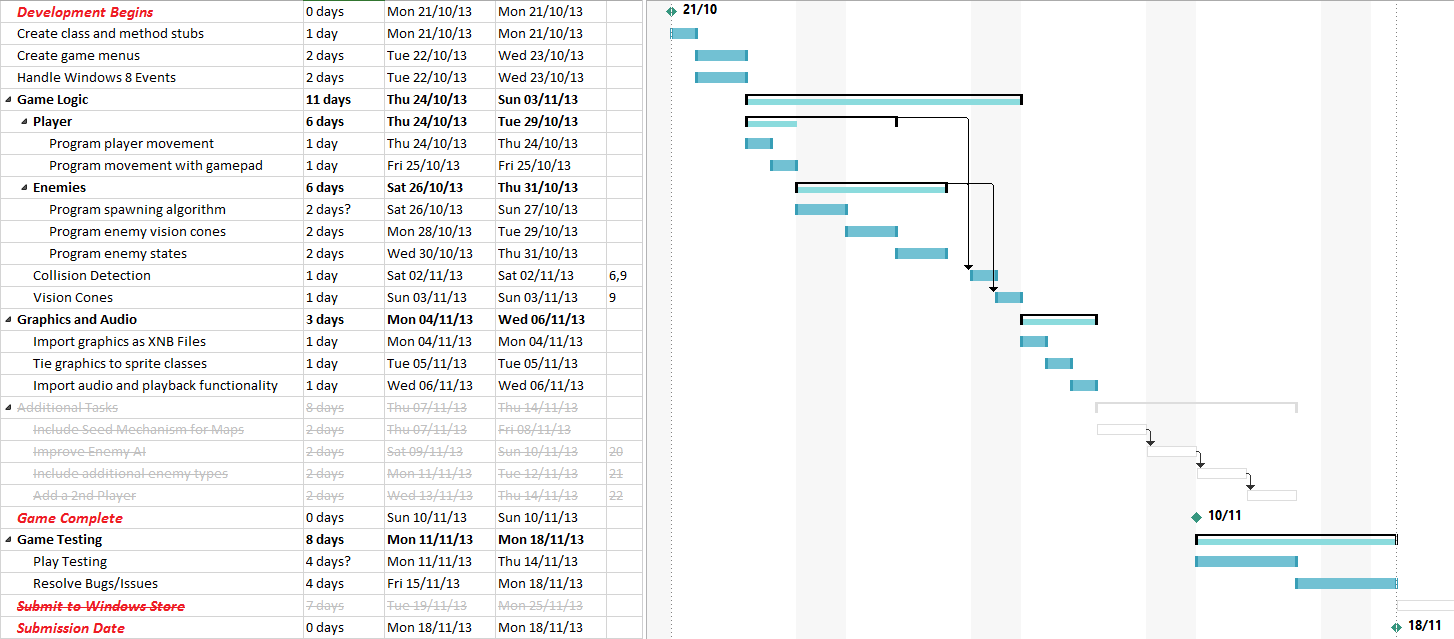
Paused

Only you can save Jacques…  
  
Resume Game

Game Options

Quit Game

# Time Management Plan

The implementation of the project will begin on the 21st of October, and is expected to last four weeks. The completion date is the 18th of November. There are several key milestones such as menu completion, enemy AI completion and random map completion. These are outlined in greater depth in the Gantt chart below.

# UML Diagrams

## Activity Diagrams

### Main Menu Activity

Game Activity



## State Transition Diagrams

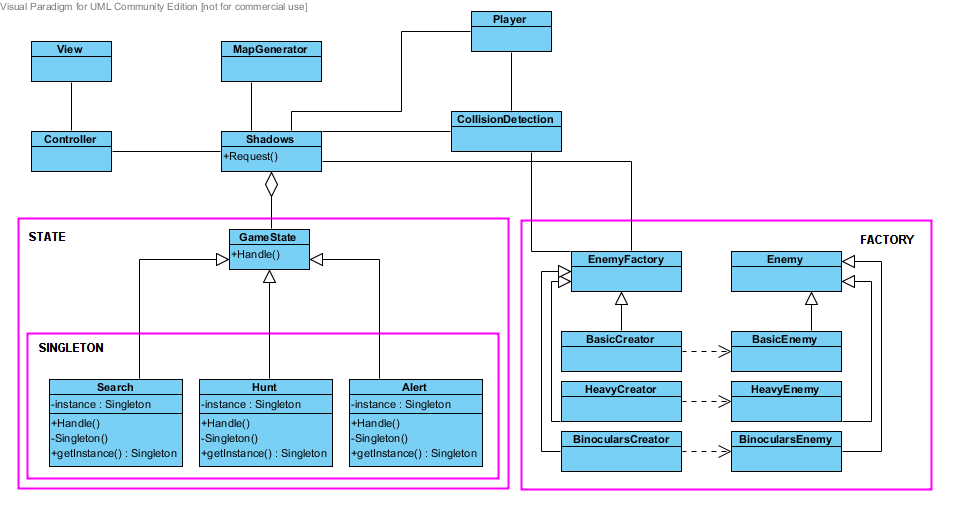
### Handling Windows 8 UI Snapping



### Player Detection



## Class Diagram



# Diagram Justification

## Behaviour Diagrams

The behaviour diagrams for Shadows have been determined by logically stepping through the process of the game, analysing the possible outcomes of each activity. The activities related to the game are fairly simple as it is (at its simplest) a game of cat and mouse. We purely need to determine what the player wishes to do in the menu activity, and if the player has been caught in the game activity.

## State Diagrams

The state diagrams for Shadows show the two main state issues relating to the game. One, handling the Windows 8 snapping problem. That is, the game can be snapped into position at the side of the screen. This would result in no real playing area. Thus we need to ensure that the gameplay pauses when this occurs, and we only let the game run when in full screen mode. In the actual implementation of the game, we will detect the percentage of the screen real estate that the game currently has, and determine if this is a satisfactory amount to play the game.

The player detection diagram shows how the game states change throughout time based on if the player has been spotted or not. We can clearly see that the enemies will begin in the search state (as they are surveying the area) and if they detect the player, they will go into hunt mode (as they are aware of the player). If the player is undetected for 30 seconds, they shall fall into the alert phase. If the player remains undetected for another 30 seconds, the game shall return to the search state. This pattern repeats until the player is caught.

## Class Diagram

The class diagram accurately portrays the expected layout of classes and design patterns within my code. The classes within the diagram show the design patterns used, and the reasoning behind these patterns is discussed in more detail in the following technical implementation section. The shadows class is the main control class of the game, and is the model in the MVC pattern. The collision detection class is required to determine when the player collides with either the enemy or objects within the level. The EnemyFactory is required in order to produce enemies of varying types using the Factory design pattern. The MapGenerator will provide a random number generator in order to create maps randomly, or create a map based on a provided seed.

# Technical Implementation

## Design Patterns

Shadows implements several design patterns such as **MVC**, **Factory**,and **State** (along with **Singleton**). The reasoning behind their individual use is discussed in greater detail below. The use of these patterns is to aid in maintainability in reusability. It is important to note that the patterns have been selected as they fit the code, not that the code has been forced to fit the pattern.

### Model View Controller (MVC)

The MVC design pattern allows us to separate our game logic, our controller (the interface between the player and the game) and our graphical display. This leads to a higher cohesion as we split away our logic from our interface and leads to reduced complexity. As we reduce the complexity, we make the code much easier to reuse and improve the flexibility.

In MonoGame, this means that we have separate methods for drawing to the screen and updating our game logic. The controller part of MVC is left up to the developer to separate and implement.

### Factory

The factory pattern is used to allow future implementation of different types of enemies, for example those with a stun-gun. The Factory pattern allows us to follow the open-closed design principle, which says that code should be open for extension but closed for modification. This makes sense as enemies will all have the same basic characteristics but some may be extended (for example, with different weapons or movement speeds). This would also allow slightly different rule sets to be applied to each type of character.

### State

The State design pattern allows us to repeat logical “states” of our program, for example, the guards going into an alert state when they will actively hunt the player (for example, after a recent detection). The states of the game are discussed in greater detail within the Detailed Game Design section of this report. The state pattern is used as it provides the clearest and simplest means of altering an objects behaviour based on a game state. It encapsulates the variation of behaviour based on the state of the game, rather than using massive conditional statements.

### Singleton

This is used as part of the **State** design pattern, as each state will only ever have a single instance. Each potential state (**Search**, **Hunt** and **Alert**) can only be instantiated once, and there will only ever be one Enemy **Factory** created. This means that at any one time, only one single state can exist. It also means that when we leave a state, we destroy the state object we created previously (and then create the new state).

# Game Features

## Collision Detection

In Shadows, the collision detection is used in a number of cases:

* Edge Detection (with the edge of the map)
* Object Collisions (walking into crates)
* Enemy Vision Cones

Clearly the game will need to implement collision detection in order to prevent the player walking over objects on the map, and stop the player walking off the edge of the screen. The collision detection would also be used to provide the enemies with a method of “seeing” the player by detecting if the player has collided with their cone of vision. This will be implemented using bounding boxes and checking for intersections.

It is important to have accurate and fast collision detection in order to not break gameplay and have a clear objective (avoid the enemy and their line of sight). Without collision detection, there would be no gameplay possible. It is possible to implement collision detection using simple rectangles, however this would soon become frustrating as the line of sight of an enemy is a cone shape. It is important to have accurate collision detection as if too lenient (the game poses no challenge) or too strict (the game is too challenging) will be to the detriment of the enjoyment of the player.

## Artificial Intelligence

Every enemy inside Shadows has a set of rules tied to them based on the current state of the game, using these rules allows the enemy to appear intelligent. We discussed the possible states of the game in the Gameplay section of this report. The rules will be similar to that of Pac-Man, where the enemies (depending on the state) may roam around the map randomly, head to tiles near you or come directly for your current position.

This method has been chosen as it shall be relatively easy to implement and will provide an enemy that will prove both challenging and clever. Although the rules are simplistic, the use of states will allow enemies to constantly change their behaviour and thus appear to be thinking. In future iterations, the AI of each enemy type could be modified due to the use of the factory class.

## Other Features

If time permits, it would be useful to implement an online leader board showing the most played seeds and the longest times. Another interesting feature would be to allow power ups that are either spawned on the ground or are awarded to the player for surviving a set amount of time. Abilities that could be granted would range from viewing enemy vision cones, to creating distractions within the map to lure enemies away from your current position.

# Play Testing

In order to ensure a high quality product is produced, play testing will be conducted throughout the development process. Every area of the game will be tested from the menus to the actual gameplay itself. This shall be conducted as objectively as possible by using testers who are outside my social circle and have no desire to muddy the results.

Players will be asked to test levels to judge the quality of the random number generator, and share seeds to ensure that maps replicate if the same seed is used again.

# Bibliography

Banas, D. (2012). *Design Patterns Video Tutorial*. Retrieved from YouTube: http://www.youtube.com/watch?v=vNHpsC5ng\_E&list=PLF206E906175C7E07

Bautista, N. (2010, July 7). *A Beginner’s Guide to Design Patterns.* Retrieved from Tuts+: http://net.tutsplus.com/articles/general/a-beginners-guide-to-design-patterns/

Credits, E. (2012). *Extra Credits: Playtesting*. Retrieved from YouTube: http://www.youtube.com/watch?v=on7endO4lPY

Freeman, E., Robson, E., Bates, B., & Sierra, K. (2004). *Head First Design Patterns.* O'Reilly Media.

Rogers, S. (2010). *Level Up!: The Guide to Great Video Game Design.* John Wiley & Sons .

Thompson, J. (2007). *The Computer Game Design Course: Principles, Practices and Techniques for the Aspiring Game Designer.* Thames and Hudson Ltd.

Whitaker, R. (2013). *MonoGame Tutorials*. Retrieved from RB Whitaker's Wiki: http://rbwhitaker.wikidot.com/monogame-tutorials

Starting Audio:  
http://www.freesound.org/people/Erokia/sounds/185239/

Crate Images:

http://opengameart.org/content/2d-wooden-box/

http://opengameart.org/content/cartoon-wooden-crate/

Sprite Sheets:

http://opengameart.org/content/141-military-icons-set/

Background Image:

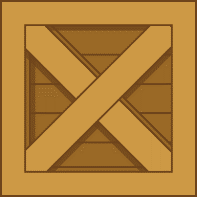
http://subtlepatterns.com/gun-metal/

Font:

http://www.misprintedtype.com/

# Appendices

## Appendix One

Vector Crate (Left), Sprite Crate (Right)

## Appendix Two

Orginal Icons and Text Images (30x30 icon, 310x150 icon and 620x300 splash screen text)



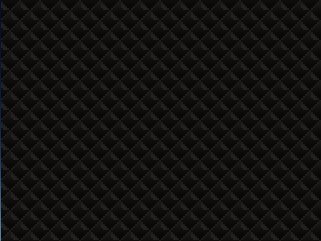
## Appendix Three

A sample of the imagery proposed by Lauren Howat



## Appendix Four

Background Image (Repeated in X and Y), Sample of Repeated background





## 

1. http://www.monogame.net/ [↑](#footnote-ref-1)
2. http://www.incognitagame.com/ [↑](#footnote-ref-2)
3. http://www.gunpointgame.com/ [↑](#footnote-ref-3)
4. http://www.sharpdx.org/ [↑](#footnote-ref-4)
5. http://monogame.net/price [↑](#footnote-ref-5)
6. http://www.opengameart.org/ [↑](#footnote-ref-6)
7. http://www.misprintedtype.com [↑](#footnote-ref-7)
8. http://www.laurenhowat.com [↑](#footnote-ref-8)
9. http://msdn.microsoft.com/en-us/library/windows/apps/hh700394.aspx [↑](#footnote-ref-9)
10. http://freesound.org/ [↑](#footnote-ref-10)
11. http://www.minecraft.net/ [↑](#footnote-ref-11)