# Downlink

Alastair Campbell, B00256757

May 2, 2014

I declare that the work submitted is my own unless otherwise stated.

## 1 Project Files

**Game.fla** The main .fla file for the project.

**Main.as** The main class containing almost all of the project code.

**circle.as** One of the code fragments used in level two.

**pentagon.as** One of the code fragments used in level two.

**square.as** One of the code fragments used in level two.

**evilcircle.as** One of the enemy (evil) code fragments used in level two.

**evilpentagon.as** One of the enemy (evil) code fragments used in level two.

**evilsquare.as** One of the enemy (evil) code fragments used in level two.

**Fragment.as** The superclass for all level four code fragments.

**Level2Player.as** Class used for the player controlled object on level two.

**LightsOut.as** Class used for the level three game (Lights out).

  **Light.as** The Light class used in the LightsOut game.

**Xerxes.as** Class used for the player on level one (maze level)

## 2 User Manual

### 2.1 Introduction

Welcome to Downlink! Downlink is very heavily based on the System Shock series and functions as a tribute and a prologue to the events of the first game, System Shock. Your role is to disable the morality functions of SHODAN on behalf of corrupt corporate executive Edward Diego.

There are four completely different levels to challenge you in different ways! Are you persistant and observant enough to get past the maze? Do you have the quick reflexes to get through the kernel core and collect the fragments before you run afoul of the corrupt code? Do you have the logical reasoning abilities to reverse the core encryption? Finally, are you quick enough to code an infinite loop into SHODAN?

Downlink also features an exciting soundtrack and animated pictures of SHODAN!

## 2.2 How to play

### 2.2.1 Level 1 - Navigate the kernel maze

Your first task is to get into the file system. Navigate through the maze using the W, S, A and D keys. The kernel security will push you out of the walls of the maze, but be careful you don't get stuck! To win this level, simply get to the center of the maze.

### 2.2.2 Level 2 - Collect the code fragments

On this level, you have entered the core of the SHODAN system. It is a very dangerous place! Beware the corrupt code fragments floating around in cyberspace - these are red! You need to collect the various fragments of code required to break SHODAN's morality functions - simply move your character into one of the green code fragments to collect it! Once you have all four, you will automatically enter into the next level.

### 2.2.3 Level 3 - Decrypt the code loop

SHODAN's central code loop is encrypted with the little-known LightsOut encryption. Luckily, there is a back-door into the system. Click the lights in the correct order to disable the core encryption. Once all of the lights are off (dark green), you will progress to the next level.

### 2.2.4 Level 4 - Program the loop

Now that you have broken in, you have limited time to crack the system before the intrusion detection system notices you – approximately 10,000 compute cycles. You need to click and drag the four code fragments into the screen in the right order. Once you have done all of this, you have won!

## 2.3 Support

You can contact our dedicated support team at `B00256757@studentmail.uws.ac.uk` for a swift response to any issue you have!

## 3 Techniques Used

## 3.1 Levels

### 3.1.1 Main Menu

The main menu was relatively simply implemented. There are three seperate keyframes, each with a page of the menu. The main menu's home page has three clickable text areas, while the other two pages (instructions and story) each have one clickable "back" button. Each clickable button has an event handler to move it to the relevant keyframe. There were no issues with implementing the main menu.

### 3.1.2 Level 1 - Maze

The maze is relatively simple. I made some rectangles within eachother and then deleted “doors” in the sides of the walls, to allow the character to pass through. The collision was more problematic - the regular method of doing a `hitTestObject(...)` on the whole thing seemed to always return true, so I had to use `hitTestPoint(...)` instead. The code used to handle collisions is also pretty strange - I wanted the character to “bounce off” of the walls rather than just not hit them, but the affect wasn’t as nice as I had hoped it would be. Additionally, it is possible to get stuck in the bottom right hand corner because of this. Other than these issues, the maze works relatively well.

### 3.1.3 Level 2 - Collect fragments

This level caused the most issues. I tried several methods (arrays, inheritance, inner classes...) to implement what I wanted to do without it being unreliable or buggy with limited success. In the end, each fragment ended up as it’s own class and object and got referred to individually. This seems to work fairly well. The main issue with this level is that sometimes you can lose as soon as you start because of the chance that an object will start in the same place as you. Additionally, this level causes issues with `gotoAndStop(...)` in that if you try to use it while the level is coded in, the application will almost always fail to run properly. Finally, if you hit the “Game Over” screen, you can’t restart. This is because of the aforementioned `gotoAndStop(...)` issue.

### 3.1.4 Level 3 - Decrypt Code

This is my best-designed level. It stores an array of lights in a very neat way and the code is quite nicely expandable - it accepts parameters for almost everything. It will happily let you play a game with a 100x100 grid with no issues. Additionally the way it flips lights by sending method calls to the parent movieclip is a very nice way to do it. In retrospect, I should have made all of my game levels seperate `.fla` files and included them the way I did with this level.

### 3.1.5 Level 4 - Compile Code

This level is also relatively simple to complete. It features a timer and drag-and-drop functionality. The timer uses milliseconds but calls them “compute cycles” to try and keep the timer relevant to the story. The way that the code compiles is also pretty neat, using a Fragment superclass with a `compileAndLink()` method.

## 3.2 Components

### 3.2.1 Sounds

Four songs play during the game relevant to the level you are playing. This uses a SoundController and Sound classes.

### 3.2.2 Movies-within-movies

The game features several animated movieclips within other movieclips. For example, the SHODAN photo prevalant throughout much of the game is an animated movieclip, the code fragments in level four are also movieClips with a separate keyframe for uncompiled and compiled states and the lights in the decryption level are also MovieClips with separate frames for on and off.

## 4 Issues

There are a few issues with this project:

- In the maze level, it is possible to get stuck in the bottom right corner.
- In the code fragment collection level, it is possible to die instantly.
- In the code fragment collection level, two objects can start in the same place and follow the same paths.
- It is not possible to restart from the game over screen.

## 5 Sources

Code was not directly copied from any source, however some resources were used as reference:

- Adobe Flash Developer Reference
- StackExchange
- Examples on Moodle