Manchester Metropolitan University

**ACADEMIC SERVICES**
**Faculty Of Science & Engineering**

# Coursework Cover Sheet

CARTER Thomas

**Instructions. 1** Print this Cover Sheet. **2** Check all the details below are correct. **3** Tick the Yes box if this is a Group Submission. **4** Write the name of the tutor who will be marking this work. **5** Tick the confirmation box. **6** Attach/include the sheet with your work. **7** Submit your work by the submission deadline to the appropriate location. **8** An email receipt will be sent to your MMU email account.

**Late Submissions.** Penalties will be incurred in accordance with the University regulations for late submission. See the MMU Student Life website for more information at www.mmu.ac.uk/student-life

**Unit Code**
6G4Z0020_2425_9

**Unit Name**
Programming (6G4Z0020_2425_9)

**Submission Deadline**
12-Jan-2025 21:00
Revised deadline

**Assignment Description**
2CWK50 - 2 Report 50% (6G4Z0020_2425_9)

**Submission ID**
Xbaac2e62

**Student Number**
23627359

**This work is a Group Submission**

☐ Yes ☑ No

**Name of Unit Leader**

Dr David Mclean

I confirm that the information above is correct and that I have read and understood the University Assessment Regulations (www.mmu.ac.uk/student-life) with regard to plagiarism. I confirm that this is all my own work.

☑ Tick box to confirm

Date Received (Office Use Only)

Generated 10-Jan-2025 22:59

Mark

S-XBAAC2E62-00095654-C

# Prisoner Escape: The Game

By Thomas Carter

Main Report Word Count: 480

# Table of Contents

# Prisoner Escape: The Game

## Game Outline

The game's outline is that you, the player have just escaped prison, but the police are after you. Your aim is to escape them for as long as you can, because even if a single police unit touches you, you are dead or captured. This also detailed to the player in game on the splash screen.

On the code side of the game, it is managed by management systems. Each of these are classes in separate scripts.

```
// Game Management Tool Object Declarations
Clock clock;              // Manages game time and provides timing functionalities.
playfieldManager playfield_Man; // Handles gamefield during game play. Object movement, Background Display etc
Player player;            // Represents the player character and its interactions.
Collision_Man collide;    // Manages collision detection between game objects.
Obstacle_Manager obstacle_Man;  // Controls the spawning and behavior of obstacles.
Difficulty difficulty;     // Manages game difficulty scaling.
highScore highscore; // Highscore management
```

The following sections show my development of the game and understanding of my code.

## Development Timeline

The development was developed in the following stages:

| Stage 1 | Stage 2 | Stage 3 | Stage 4 |
|---|---|---|---|
| Player and obstacle display | - Spawning of obstacles<br>- Management of obstacles and other items on screen | Various other game mechanics examples:<br>- Clock<br>- High score<br>- etc | UI:<br>- Screen objects and screen elements. |

## Game Mode

An enum of different game modes manages the operation of the game. This dictates what is currently being displayed in each frame in draw() method. Both the enum and switch method are shown below.

1. Game mode enum (Source: gameMode.pde ):

```
// Enum to define different game states
enum gameMode {
  SPLASH,
  START,
  PAUSE,
  PLAYING,
  DEATH
}
```

2. Switch controller (Source: main.pde ):

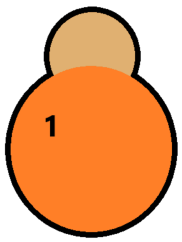
```
void draw()
{
  // Handle different game modes based on the current state (splash screen, start, playing, pause, or death).
  // The gamemode variable determining which screen or gameplay state is active at any given time.
  switch(gamemode) {
    case SPLASH:
        // Display splash screen when game intialised
        splash.Display();
        break;
    case START:
        // Display start screen
        start.Display();
        break;
    case PLAYING:
        // Handle game logic and systems when game is active/unpaused

        clock.Run(); // Run the game clock
        difficulty.Play(); // Adjust game difficulty over time

        background(255); // Clear the screen

        playfield_Man.display(); // Display game environment and manage on screen objects
        player.display(); // Display Player character
        obstacle_Man.Display(); // Display Obstacles

        obstacle_Man.Spawner(); // Spawn obstacles
        obstacle_Man.RunLoopRun(); // Execute obstacle specified behaviour

        collide.checkCollisions(collidables); // Check for collisions between all collidables
        player_HUD.Display(); // Show player HUD
        break;
    case DEATH:
        // Display the death screen
        death.Display();
        break;
    case PAUSE:
        // Display pause screen
        pause.Display();
        break;
  }
}
```

## Player

The player is controlled by Player_Manager.pde. The original look of the player was a box fixed to the middle by an inbuilt grid system (see script).

```
// First player appearance
//fill(243,125,61);
//rect(playerx, playery, playersize,playersize);
```

The player now appears as a prisoner (below), using the sprite animation system built for the project and rotating by 45 degrees, depending on the direction of movement.




The keyboard cluster controls player input:

```
// Handles Player input logic
void OnKeyPressed()
{
  // Dependent on key pressed and whether action is allowed set pressed vale to true and play relevant animation
  if (keyCode == LEFT && canMoveLeft)
  {
    leftPressed = true;
    thisSprite.PlayAnimation("Left");
  }
  else if (keyCode == RIGHT && canMoveRight)
  {
    rightPressed = true;
    thisSprite.PlayAnimation("Right");
  }
  else if (keyCode == UP)
  {
    upPressed = true;
    thisSprite.PlayAnimation("Up");
  }
  else if (keyCode == DOWN)
  {
    downPressed = true;
  }
  else if (key == 'p' || key == 'P') // Pause game
  {
    clock.Pause(); // Pauses game clock
    gamemode = gameMode.PAUSE; // set game state to pause
  }

  UpdateMovement(); // Update directional info stored based on most recent key press
}
```

## Obstacles

There are several different obstacles or police units. Each has different behaviour each. Like the player first took on a basic form as squares. Now, it also take on a different appearance dependent on unit type with a range of animation states depend on direction and action. Spawning mechanic works from each kind having a probabity of spawning and random selection. The further progress made in the more different types will spawn.

Obstacles / police units:

- Police Officer
- Police Blockade
- Armed Police
- Combined Blockade and Armed Police
- K9 Unit
- Police Car

1. Original officer look

```
// original officer display
//PFont bold = createFont("Arial-BoldMT", 128);
//fill(0);
//rectMode(CENTER);
//rect(x,y,Width,Height);
//fill(255);
//textFont(bold);
//textAlign(CENTER);
//textSize(15);
//text("P",x,y);
```

2. Two different Officer looks.

## Management Systems

The game is built around three central management systems:

1. Player Managers (Player_Manager.pde):
2. Obstacle Manager (Obstacle_Manager.pde): Each obstacle has to extend the Obstacle superclass. This script then manages spawn mechanics and movement relevant to the screen so that all move in sync.
3. Collision Manager (Collison.pde): If the object is to be collidable, it must implement a collidable interface. This script managers collision checks between objects calling collide with functions so relevant actions can be performed.

Other Systems used, check scripts for functions and behaviour:

- Animation System: Sprite.pde and Animation.pde
- Screen System: Screen.pde and Screen Elements.pde
- Tracker System: Tracker.pde
- Difficulty : Difficulty.pde. Controls games difficulty as the game progresses.

# Art References

The references for art use are as follows:

- Prisoner: Self-made.
- Officer 1, 2 and Armed Police: https://craftpix.net/freebies/2d-game-police-character-free-sprite-sheets/?srsltid=AfmBOorL6QD8p-ybpayggkLOU5O49tZ71TYtOWZ6sGA84rzHExJFGbm_
- K9: https://admurin.itch.io/top-down-mobs-dog
- Police Car: https://opengameart.org/content/top-down-pixel-police-car
- Barbed Wire: https://www.freepik.com/free-vector/metal-steel-barbed-spiral-wire-with-thorns-spikes_5723555.htm#fromView=keyword&page=1&position=0&uuid=f4db365c-d068-42d2-a610-34d72a70829f

# Mark Scheme Pointers

| Base Mark | Coding concepts required | Where? |
|---|---|---|
| 40% | **All** of the following to Pass (40%):<br>☐ Minimum 1 class (e.g. Obstacle, Player)<br>☐ Keyboard arrow cluster | **Classes:**<br>All Scripts<br>**Minimum of 1 Class:**<br>Refer to 60% to 70% grade bound below for list of obstacle classes. In addition all scripts include the use of class.<br>**Keyboard Arrow Cluster:**<br>Input handled by player manager. Uses keyboard cluster |
| 40-50% | ☐ Minimum 2 Classes (Player,Obstacle)<br>☐ Obstacle collides with the Player<br>☐ Working Boolean Collision *function* method(s)<br>☐ Obstacle objects use PImage | **Collision:**<br>Collision between object done through collidable interface and collidable manager. Boolean function found in Collision Manager Lines( - )<br>**Obstacle use of PImage:**<br>Obstacle display a PImage through animation system consisting of scripts: Sprite and Animation. |
| 50-60% | ☐ Obstacles have their own movement<br>☐ An ArrayList (or array) of Obstacle objects<br>☐ objects removed from memory (set to **null**, or removed from array/arrayList) | **Independent Obstacle Movement:**<br>Each Obstacle have there own movement using the Move() of the obstacle super class. Default definition found in Obstacle script Lines ( - )<br>**Arraylist of obstacles:**<br>In Obstacle_Manager all Obstacles are stored in array list called, |

| | | |
|---|---|---|
| | | 'obstacle_OnScreen' with 'toAdd' and 'toRemove' to manage safe addition and removal of obstacles from list. AddObstacle()and RemoveObstacle() |
| 60-70% | ☐ 2nd Class (different type) of Obstacle<br>☐ Class-inheritance<br><br>☐ Player changes appearance dependent on arrow key | **Different Types of Obstacles:**<br>The different obstacle types are listed below, each have there own class and script:<br>• Police Officer<br>• Police Blockade<br>• Armed Police<br>• Combined Blockade and Armed Police<br>• K9 Unit<br>• Police Car<br><br>**Class Inheritance:**<br>Class inheritance is exhibited in the following classes:<br><br>• All screen objects. 4 screens inherit Screen class.<br><br>• The following inherit screenObject class:<br>○ Obstacle<br>○ Player<br>○ Road<br><br>• The following inherit Obstacle class:<br>○ Police Officer<br>○ Police Blockade<br>○ Collision_Block<br>○ Armed Police<br>○ Combined Blockade and Armed Police<br>○ K9 Unit<br>○ Police Car<br>○ Projectile<br><br>• The following inherit Screen:<br>○ Splash Screen<br>○ Start Screen<br>○ Pause Screen<br>○ Death Screen |

| | | |
|---|---|---|
| | | • The following inherit Screen Element:<br>  ○ Text<br>  ○ Button |
| 70-80% | □ Animation sequences<br>□ File handling – high score(s) saved and read from file<br><br>□ polymorphism with array/arrayList | **Animations:**<br>Animations sequences are exhibited through the Animation and Sprite classes. Sprite controlling and holding animatiosn relevant to the object it is attached to.<br><br>**File handling:**<br>File handling is shown through the saving, loading and display of high score. See highScore.pde<br><br>**Poly Morphism:**<br>Poly morphism is exhibited through a number of different scripts. The main being through array list of obstacles in obstacle manager and screen objects in screen manager. |
| 80%+ | □ Use of an Interface or abstract class | **Interfaces:**<br>The following interfaces are used:<br>• Collidable in collision.pde<br><br>**Abstract Classes:**<br>The following abstract classes are used:<br>• Screen. Check Screen.pde |