The University of Melbourne
School of Computing and Information Systems
SWEN20003 Object Oriented Software Development

# ShadowMario

**Project 1, Semester 1, 2024**
Released: Monday, 25th March 2024 at 11:30pm AEST
Initial Submission Due: Thursday, 28th March 2024 at 11:30pm AEST
Project Due: Wednesday, 17th April 2024 at 11:30pm AEDT

**Please read the complete specification before starting on the project, because there are important instructions through to the end!**

## Overview

Welcome to the first project for SWEN20003, Semester 1, 2024. Across Project 1 and 2, you will design and create an arcade game in Java called *ShadowMario* (an adaptation of the original 90's arcade game *Super Mario Bros*). In this project, you will create the first level of the full game that you will complete in Project 2B. This is an **individual project**. You may discuss it with other students, but all of the implementation must be your **own work**. By submitting the project you declare that you understand the University's policy on academic integrity and aware of consequences of any infringement, including the use of **artificial intelligence**.

You may use any platform and tools you wish to develop the game, but we recommend using IntelliJ IDEA for Java development as this is what we will support in class.

The purpose of this project is to:

- Give you experience working with an object-oriented programming language (Java),
- Introduce simple game programming concepts (2D graphics, input, simple calculations)
- Give you experience working with a simple external library (Bagel)

**Extensions & late submissions:** If you need an extension for the project, please complete the Extension form in the **Projects** module on Canvas. Make sure you explain your situation with some supporting documentation such as a medical certificate, academic adjustment plan, wedding invitation, etc. You will receive an email saying if the extension was approved or if we need more information.

If you submit late (**either** with or without an extension), please complete the Late form in the **Projects** module on Canvas. For both forms, you need to be logged in using your **university** account. Please **do not** email any of the teaching team regarding extensions or late submissions. All of this is explained again in more detail at the end of this specification.

You **must** make at least 5 commits (excluding the Initial Submission commit) throughout the development of the project, and they must have meaningful messages. This is also explained in more detail at the end of the specification.

### *Game Overview*

*"The aim is simple - move the **player** to jump over the **enemies** and collect the **coins**. Can you reach the **end flag** to complete the level?*

The game consists of three levels - Project 1 will only feature the first level. The player has to press the arrow keys to move left, right or jump. The player can collect coins by colliding with them - collecting one coin, increases the score by one. The player can avoid the stationary enemies by jumping over them (*the enemies will be moving in Project 2!*). If the player collides with an enemy, they will lose health points. To win, the player needs to reach the end flag. If the player's health points reduce to zero, the game ends.

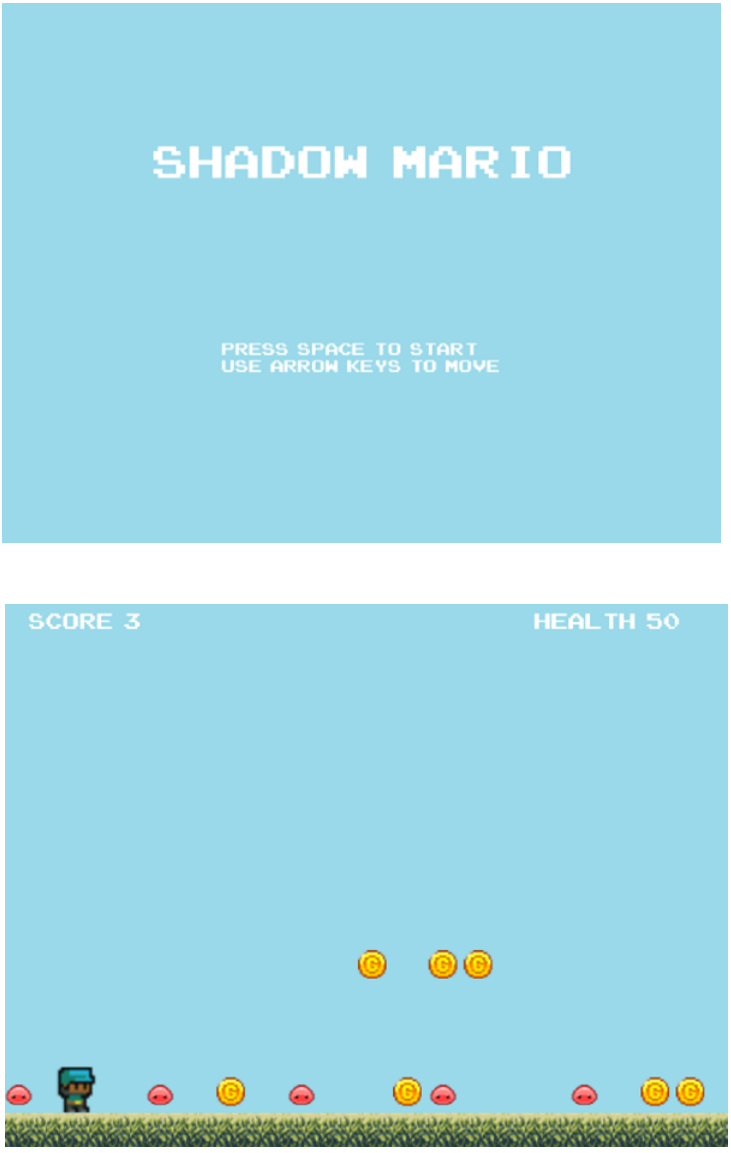


Figure 1: Completed Project 1 Screenshot

# The Game Engine

The **B**asic **A**cademic **G**ame **E**ngine **L**ibrary (Bagel) is a game engine that you will use to develop your game. You can find the documentation for Bagel here.

### *Coordinates*

Every coordinate on the screen is described by an $(x, y)$ pair. $(0, 0)$ represents the top-left of the screen, and coordinates increase towards the bottom-right. Each of these coordinates is called a *pixel*. The Bagel `Point` class encapsulates this.

### *Frames*

Bagel will refresh the program's logic at the same refresh rate as your monitor. Each time, the screen will be cleared to a blank state and all of the graphics are drawn again. Each of these steps is called a **frame**. Every time a frame is to be rendered, the `update()` method in `ShadowMario` is called. It is in this method that you are expected to update the state of the game.

The refresh rate is typically `120` times per second (Hz) but some devices might have a lower rate of 60Hz. In this case, when your game is running, it may look different to the demo videos as the constant values in this specification have been chosen for a refresh rate of 120Hz. For your convenience, when writing and testing your code, you **may** change these values to make your game playable (these changes are explained later). If you do change the values, **remember** to change them back to the original specification values before submitting, as your code will be **marked on 120Hz screens.**

# The Game Elements

Below is an outline of the different game elements you will need to implement.

### *Window and Background*

The background (`background.png`) should be rendered on the screen and completely fill up your window throughout the game. The default window size should be `1024 * 768` pixels. The background has already been implemented for you in the skeleton package.

### *Messages*

All messages should be rendered with the font provided in the `res` folder (`FSO8BITR.ttf`). If not otherwise specified, message coordinates should be roughly centered.

**Hint:** The `drawString()` method in the Font class uses the given coordinates as the bottom left of the message. So to center the message, you will need to calculate the coordinates using the `Window.getWidth() and Window.getHeight()` methods.

### *Game Start*

When the game is run, a title message that reads `SHADOW MARIO` should be rendered in the font provided. The bottom left corner of this message should be located at `(220, 250)`, in size `64`.

Additionally, an instruction message consisting of 2 lines:

```
PRESS SPACE TO START
USE ARROW KEYS TO MOVE
```

should be rendered **below** the title message, in the font provided, in size `24`. The bottom left of the first line in the message should be calculated as follows: the x-coordinate should be centered horizontally and the y-coordinate should be at `500` pixels.

There must be **adequate spacing** between the 2 lines to ensure readability (you can decide on the value of this spacing yourself, as long as it's not small enough that the text overlaps or too big that it doesn't fit within the screen).

### *Properties File*

The key values of the game are listed in two **properties files** which are given in the skeleton package. The message coordinates, image filenames and other values are given in the `app.properties` file. The message strings are given in the `message_en.properties` file. These files **shouldn't** be edited (unless you need to adjust values for any frame rate issues).

To read a value from one of these properties, a `Properties object` must be created. The `getProperty` method can be called on this object with the required value given as the parameter. For your reference, the skeleton package contains an example of how to read the background image filename, window width and window height values.

### *World File*

The entities will be defined in a **world file**, describing the type and their position in the window. The world file is located at `res/level1.csv`. A world file is a comma-separated value (CSV) file with rows in one of the following formats:

```
Type of entity, x-coordinate, y-coordinate
```

An example of a world file:

```
PLATFORM,3000,745
PLAYER,100,687
COIN,300,510
ENEMY,400,695
END_FLAG,4100,670
```

The given `(x, y)` coordinates refer to the centre of each image and these coordinates should be used to draw each image. You must actually load it—copying and pasting the data, for example, is not allowed. Marking will be conducted on a hidden **different** CSV file of the same format. **Note:**

You can assume that there are 50 lines in the CSV and that there will always be at least one of each for all the entities. You can also assume there are 14 enemies and 33 coins for Project 1.

### *Win Conditions*

If the player reaches (**collides with**) the end flag, this is considered as a win. A winning message, that consists of 2 lines:

```
CONGRATULATIONS, YOU WON!
PRESS SPACE TO CONTINUE
```

should be rendered, in the font provided, in size `24`. The bottom left of the first line in the message should be calculated as follows: the x-coordinate should be centered horizontally and the y-coordinate should be at `400` pixels. If the space key is pressed, the game returns to the start screen and allows the player to play again.

### *Lose Conditions*

If the player's health points reduces to **0 or below**, this is considered as a loss and the game ends. A message, that consists of 2 lines:

```
GAME OVER, YOU LOST!
PRESS SPACE TO CONTINUE
```

should be rendered, in the font provided, in size `24`. The bottom left of the first line in the message should be calculated as follows: the x-coordinate should be centered horizontally and the y-coordinate should be at `400` pixels.

Once again if the space key is pressed, the game returns to the start screen and allows the player to play again. If the player terminates the game window at any point (by pressing the Escape key or by clicking the Exit button), the window will simply close and no message will be shown.

### *Game Entities*

The following game entities have an associated image (or multiple!) and a starting location `(x, y)`. Remember that all images are drawn from the centre of the image using these coordinates.

#### Player

In our game, the player can move on screen in one of three directions (left, right and up) when the corresponding arrow key is pressed. However, for our ease of implementation, we assume the player can **only** move vertically and remains stationary in the horizontal direction (i.e. the other entities will be moving in relation to the player's arrow key pressed - *this is explained in detail later*).

(a) player_left.png

(b) player_right.png

Figure 2: The player's images

The player is represented by the two images shown above. Based on the direction the player is moving, the **corresponding** image should be rendered. The player will start the game facing right.

The player's jumping upwards motion will be considered in 3 stages, where the speed in the vertical direction will change:

- Player is currently on platform & up arrow key is pressed => the vertical speed should be set to `-20` (i.e. the y-coordinate will be decreasing by `20 pixels per frame`).
- During the player's jumping motion => vertical speed should increase by `1` each frame.
- Player has finished jump & has reached platform again => vertical speed should be set to 0 and the player **should not** move below the platform.

**Hint:** Remember that y increases in the downward direction on screen.


Figure 3: Player's score

The player has an associated **score**. When the player collides with a coin, the player's score increases by `1` (the points value of the coin). The score is rendered in the top left corner of the screen in the format of `"SCORE k"` where `k` is the current score. The bottom left corner of this message should be located at `(35, 35)` and the font size should be 30.




Figure 4: Player's health

When a player collides with an enemy, the player's **health** decreases by `0.5` (the damage points value of the enemy). The player starts the game with a health value of `1`. The health value is rendered in the top right corner of the screen in the format of `"HEALTH k"` where `k` is the current health, shown as integer percentage of the total health. The bottom left corner of this message should be located at `(750, 35)` and the font size should be 30.

If the player's health value becomes less than or equal to zero, the player moves vertically down off the screen and the game ends. This is done by setting the vertical speed to **2 pixels per frame**.

## Enemy

Figure 5: enemy.png

An enemy is an entity shown by `enemy.png`, that can move in the **horizontal** direction. It has a damage points value of `0.5`. When the player's arrow keys are pressed, the enemy will move accordingly as described below.

When the player's *right* arrow key is pressed or held down, the enemy will move to the *left* by **5 pixels per frame**. When the player's *left* arrow key is pressed or held down, the enemy will move to the *right* by the same speed. When the enemy collides with a player, it inflicts damage to the player's health as described earlier. Once an enemy has inflicted damage once, it cannot inflict damage again even if there are further collisions.

### Collision Detection

To detect collisions, a **range** is first calculated by adding the radius of the enemy image and the radius of the player image. Both values are given in the `app.properties` file. The **current distance** between the player and the enemy is determined by calculating the **Euclidean distance** between the two `(x, y)` coordinates. If the current distance is **less than or equal to** the range, this is considered as a collision.

## Coin

A coin is an entity shown by `coin.png`, that can move in **both** horizontal and vertical directions. It has a points value of `1`. When the player's arrow keys are pressed, the coin will move as described below.

Figure 6: coin.png

When the player's *right* arrow key is pressed or held down, the coin will move to the *left* by **5 pixels per frame**. When the player's *left* arrow key is pressed or held down, the coin will move to the *right* by the same speed.

When a coin collides with a player, the player's score increases by `1`. The collision detection is determined in the same way as described above in the *Enemy* section. Once a collision has happened, a coin will move upwards and disappear off screen. This is done by setting the vertical speed to **-10 pixels per frame**.

## Platform

Figure 7: platform.png (cropped to show on one page)

The platform is an entity shown by `platform.png`, that can move in the **horizontal** direction. When the player's arrow keys are pressed, the platform will move as described below.

When the player's *right* arrow key is pressed or held down, the platform will move to the *left* by **5 pixels per frame**. When the player's *left* arrow key is pressed or held down, the platform will move to the *right* by the same speed, only if the platform's current x-coordinate is **less than 3000**.

**End Flag**

The end flag is an entity shown by `endflag.png`, that can move in the **horizontal** direction. When the player's arrow keys are pressed, the flag will move as described below.

Figure 8: endflag.png

When the player's *right* arrow key is pressed or held down, the flag will move to the *left* by **5 pixels per frame**. When the player's *left* arrow key is pressed or held down, the flag will move to the *right* by the same speed.

When the flag collides with a player, the game ends as a **win** for the player. The collision detection is checked in the same way as described in the *Enemy* section.

## Your Code

You must submit a class called `ShadowMario` that contains a `main` method that runs the game as prescribed above. You may choose to create as many additional classes as you see fit, keeping in mind the principles of object oriented design discussed so far in the subject. You will be assessed based on your code running correctly, as well as the effective use of Java concepts. As always in software engineering, appropriate comments and variables/method/class names are important.

## Implementation Checklist

To get you started, here is a checklist of the game features, with a suggested order for implementing them:

- Draw the title and game instruction messages on screen
- Read the world file, and draw the platform on screen
- Draw the player, one coin, one enemy and the platform on screen
- Implement movement logic for the 4 entities above
- Implement image rendering and movement for all the entities in the world file
- Implement the scoring behaviour when the player collides with a coin
- Implement the health logic for the player
- Implement win detection and draw winning message on screen
- Implement lose detection and draw losing message on screen

## Supplied Package

You will be given a package called `project-1-skeleton.zip` that contains the following: (1) Skeleton code for the `ShadowMario` and `IOUtils` classes to help you get started, stored in the `src`

folder. (2) All graphics, fonts and properties that you need to build the game, stored in the `res` folder. (3). The `pom.xml` file required for Maven. Here is a more detailed description:

- `res/` – The graphics and font for the game (you are not allowed to modify any of the files in this folder).
  - `background.png`: The image to represent the background.
  - `coin.png`: The image to represent a coin.
  - `endflag.png`: The image to represent the end flag.
  - `enemy.png`: The image to represent an enemy.
  - `platform.png`: The image to represent the platform.
  - `player_left.png`: The image to represent the player facing left.
  - `player_right.png`: The image to represent the player facing right.
  - `app.properties`: The properties file containing coordinates, values and file paths.
  - `message_en.properties`: The properties file containing the message strings.
  - `FSO8BITR.ttf`: The font to be used throughout this game.
  - `level1.csv`: The world file for the first level.
  - `credit.txt`: The file containing credit for the font and images (you can ignore this file).
- `src/` – The skeleton code for the game.
  - `ShadowMario.java`: The skeleton code that contains an entry point to the game and an `update()` method that draws the background.
  - `IOUtils.java`: The skeleton code that has an empty method to read a CSV file and a completed method to read a Properties file.
- `pom.xml`: File required to set up Maven dependencies.

## Submission and Marking

### Initial Submission

To ensure you start the project with a correct set-up of your local and remote repository, you must complete this Initial Submission procedure on or before **Thursday, 28th March 2024 at 11:30pm**.

1. Clone the `[user-name]-project-1` folder from GitLab.
2. Download the `project-1-skeleton.zip` package from Canvas, under Project 1.
3. Unzip it.
4. Move the **contents** of the unzipped folder to the `[user-name]-project-1` folder **in your local machine**.

5. Add, commit and push this change to your remote repository with the commit message `"initial submission"`.
6. Check that your push to Gitlab was successful and to the correct place.

After completing this, you can start implementing the project by adding code to meet the requirements of this specification. Please remember to add, commit and push your code regularly with meaningful commit messages as you progress.

You **must** complete the Initial Submission following the above instructions by the due date. Not doing this will incur a **penalty of 3 marks** for the project. It is best to do the Initial Submission before starting your project, so you can make regular commits and push to Gitlab since the very start. However, if you start working on your project locally before completing Initial Submission, that is fine too, just make sure you move all of the contents from your project folder to `[user-name]-project-1` in your local machine.

### Technical requirements

- The program must be written in the Java programming language.
- Comments and class names must be in English **only**.
- The program must not depend upon any libraries other than the Java standard library and the Bagel library (as well as Bagel's dependencies).
- The program must compile fully without errors.

Submission will take place through GitLab. You are to submit to your `<username>-project-1` repository. At the **bare minimum** you are expected to follow the structure below. You **can** create more files/directories in your repository if you want.

```
username-project-1
├── res
│   └── resources used for project 1
└── src
    ├── ShadowMario.java
    └── other Java files
```

On 17th April 2024 at 11:30pm, your latest commit will automatically be harvested from GitLab.

### Commits

You are free to push to your repository post-deadline, but only the latest commit on or before 17th April 2024 11:30pm will be marked. You **must** make at least 5 commits (excluding the Initial Submission commit) throughout the development of the project, and they must have meaningful messages (commit messages must match the code in the commit). If commits are anomalous (e.g. commit message does not match the code, commits with a large amount of code within two commits which are not far apart in time) you risk penalization.

Examples of **good, meaningful** commit messages:

- implemented movement logic
- fix the player's scoring behaviour
- refactored code for cleaner design

Examples of **bad, unhelpful** commit messages:

- fesjakhbdjl
- yeah easy finished the logic
- fixed thingzZZZ

### Good Coding Style

Good coding style is a contentious issue; however, we will be marking your code based on the following criteria:

- You should not go back and comment your code after the fact. You should try to comment as you go.
- You should be taking care to ensure proper use of visibility modifiers. Unless you have a very good reason for it, all instance variables should be private (apart from constants).
- Any constant should be defined as a final static variable (**Note:** for Image and Font objects, use only final). Constants can be public depending on usage. Don't use magic numbers!
- Think about whether your code is written to be easily extensible via appropriate use of classes.
- Make sure each class makes sense as a cohesive whole. A class should have a single well-defined purpose, and should contain all the data it needs to fulfil this purpose.

### Extensions and late submissions

If you need an **extension** for the project, please complete the Extension form in the **Projects** module on Canvas. Make sure you explain your situation with some supporting documentation such as a medical certificate, academic adjustment plan, wedding invitation, etc. You will receive an email saying if the extension was approved or if we need more information.

The project is due at **11:30pm sharp**. Any submissions received past this time (from 11:30pm onwards) will be considered late unless an extension has been granted. There will be no exceptions. There is a penalty of 1 mark for a late project, plus an additional 1 mark per 24 hours. If you submit **late** (*either* with or without an extension), please complete the Late form in the **Projects** module on Canvas. For both forms, you need to be logged in using your **university** account. Please **do not** email any of the teaching team regarding extensions or late submissions.

### Marks

Project 1 is worth **10** marks out of the total 100 for the subject. Although you may see how inheritance can be used for future extensibility, you are **not required** to use inheritance in this project.

- **NOTE:** Not completing the Initial Submission (described in the Submission and Marking section here) before beginning your project will result in a **3 mark penalty**!
- Features implemented correctly – **6.5 marks**
  - Starting screen is implemented correctly: (**0.5 marks**)
  - The player images and movement are implemented correctly: (**1 mark**)
  - The platform image and movement is implemented correctly: (**1 mark**)
  - The enemy image and movement (including player collision logic) is implemented correctly: (**1 mark**)
  - The coin image and movement (including player collision logic) is implemented correctly: (**1 mark**)
  - The end flag image and movement is implemented correctly: (**1 mark**)
  - Win detection is implemented correctly with winning message rendered: (**0.5 marks**)
  - Loss detection is implemented correctly with game-over message: (**0.5 marks**)
- Code (coding style, documentation, good object-oriented principles) – **3.5 marks**
  - Delegation and Cohesion – breaking the code down into appropriate classes, each being a complete unit that contain all their data: (**1 mark**)
  - Use of Methods – avoiding repeated code and overly long/complex methods: (**1 mark**)
  - Code Style – visibility modifiers, consistent indentation, lack of magic numbers, commenting, etc. : (**1.5 marks**)