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

ShadowDimension

Project 1, Semester 2, 2022
Released: Friday, 26/08/2022 at 8:59pm AEST
Initial Submission Due: Thursday, 01/09/2022 at 8:59pm AEST
Project Due: Friday, 09/09/2022 at 8:59pm AEST

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 2, 2022. Across Project 1 and 2, you will design and create a fantasy role-playing game in Java called *ShadowDimension* (inspired by *Stranger Things*). 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.

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)

Note: If you need an extension for the project, please complete this form. 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 this form. 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.

Game Overview

*"A dark evil has arrived in your hometown. A group of government scientists have opened a gate in their laboratory to another dimension called the **Over Under**. The Over Under is ruled by **Navec** (an evil creature of immense power) and his henchmen are called demons. The scientists thought they could control these creatures but alas they failed and are being held captive in the Over Under. Navec has created **sinkholes** that will destroy the lab and is planning on eventually destroying your world.*

*The **player**'s name is **Fae**, the daughter of one of the scientists. In order to save your father and your town, you need to avoid the sinkholes, find the gate in the lab and defeat Navec & his demons in the Over Under. . . "*

Project 1 will only feature the first level - finding the gate in the lab. The player will be able to control Fae who has to move around the walls and avoid any sinkholes that are in the lab. If the player falls into a sinkhole, the player will lose health points. To win, the player has to get to the gate, located in the bottom right of the window. If the player's health reduces to 0, the game ends.

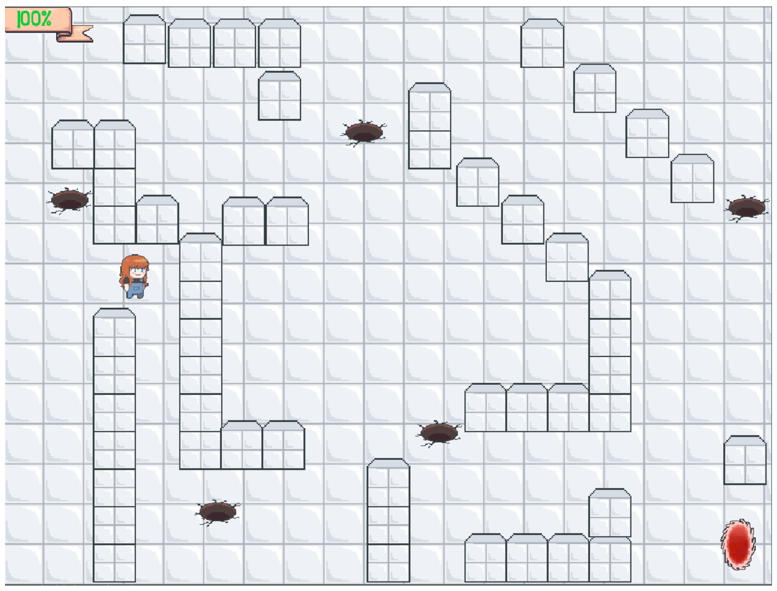


Figure 1: Completed Project 1 Screenshot

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 `ShadowDimension` is called. It is in this method that you are expected to update the state of the game.

The refresh rate is typically `60` times per second (Hz) but newer devices might have a higher rate. 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 60Hz. For your convenience, when writing and testing your code, you **may** either change these values to make your game playable or lower your monitor's refresh rate to 60Hz. 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 60Hz screens.**

Collisions

It is sometimes useful to be able to tell when two images are *overlapping*. This is called **collision detection** and can get quite complex. For this game, you can assume images are rectangles and that the player falling into a sinkhole or overlapping with a wall is a collision. Bagel contains the `Rectangle` class to help you.

The Game Elements

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

Window and Background

The background (`background0.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 `res` folder, in size `75` (unless otherwise specified). All messages should be centered both horizontally and vertically (unless otherwise specified).

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()`, `Window.getHeight()` and `Font.getWidth()` methods, and also the font size.

Game Start

When the game is run, a title message that reads `SHADOW DIMENSION` should be rendered in the font provided. The bottom left corner of this message should be located at `(260, 250)`.

Additionally, an instruction message consisting of 2 lines:

```
PRESS SPACE TO START
USE ARROW KEYS TO FIND GATE
```

should be rendered **below** the title message, in the font provided, in size `40`. The bottom left of the first line in the message should be calculated as follows: the x-coordinate should be `90` pixels below and the y-coordinate `190` pixels below.

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). Nothing else, not even the background, should be rendered in the window before the game has begun.

World File

The player, the walls and the sinkholes will be defined in a **world file**, describing the type and their position in the window. The world file is located at `res/level0.csv`. A world file is a comma-separated value (CSV) file with rows in the following format:

```
Type, x-coordinate, y-coordinate
```

An example of a world file:

```
Player,5,696
Wall,120,700
Wall,120,650
Sinkhole,255,655
```

You must actually load it—copying and pasting the data, for example, is not allowed. **Note:** You can assume that the player is always the first entry in the file and that this world file has a maximum of 60 entries.

Bounds

This is a rectangular perimeter that represents the edges of the game, which will be provided in the CSV file (`TopLeft` for the top-left `(x, y)` coordinate of the perimeter, `BottomRight` for the bottom-right). You can assume that all entities provided will have a starting location within this perimeter. The player should not be able to move **outside of this perimeter**, they should simply remain at the position they were at when they tried to cross this perimeter until the player is moved in a different direction.

Win Conditions

Once the player reaches the gate, this is considered as a win. To reach the gate, the player's `x` coordinate must be greater than or equal to 950 and the player's `y` coordinate must be greater than or equal to 670. A winning message that reads `CONGRATULATIONS!` should be rendered as described earlier in the *Messages* section. Note that nothing else (not even the background) must be displayed in the window at this time.

Lose Conditions

If there is no win, the game will continue running until it ends. As described earlier, the game can only end if the player's health points reduce to 0. A message of `GAME OVER!` should be rendered as described earlier in the *Messages* section. Note that nothing else (not even the background) must be displayed in the window at this time. 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)` on the map which are defined in the CSV file provided to you. Remember that you can assume the provided images are rectangles and make use of the `Rectangle` class in Bagel; the provided `(x, y)` coordinates for a given entity should be the **top left** of each image.

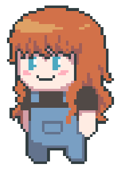
Hint: Image has the `drawFromTopLeft` method and `Rectangle` has the `intersects` method for you to use, refer to the Bagel documentation for more info.

The Player

In our game, the player is represented by Fae. The player is controlled by the arrow keys and can move continuously in one of four directions (left, right, up, down) by **2 pixels per frame** whenever an arrow key is **held down**.

The player has **health points**, which is an integer value that determines their current amount of health. The player will always start the game with the maximum number of health points, which is **100**. When receiving damage by overlapping with a sinkhole, the player will lose health points (based on the sinkhole's own damage points, as described later). Note that the player **cannot** inflict damage to a wall or a sinkhole.

If the player's health points reduce to 0, the game ends. The player's health points **do not** become negative.


(a) faeLeft.png

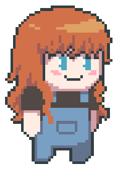

(b) faeRight.png

Figure 2: The player's images

As shown above, when moving left, `faeLeft.png` should be rendered and when moving right, `faeRight.png` should be rendered. Initally, the player will start by facing right.

Health Bar

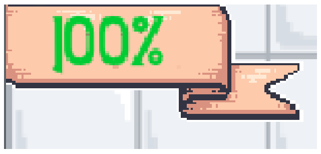


Figure 3: Health Bar

The player's current health points value is displayed as a percentage of the maximum health points on screen. This can be calculated as $\frac{CurrentHealthPoints}{MaximumHealthPoints}$. For example, if the player's current Health Points is 15 and their maximum is 20, the percentage is 75%. This percentage is rendered in the top left corner of the screen in the format of `k%` where `k` is rounded to the nearest integer. The bottom left corner of this message should be located at `(20, 25)` and the font size should be 30. Initally, the colour of this message should be green `(0, 0.8, 0.2)`. When the percentage is below 65%, the colour should be orange `(0.9, 0.6, 0)` and when it is below 35%, the colour should be red `(1, 0, 0)`.

Hint: Refer to the DrawOptions class in Bagel and how it relates to the **`drawString()`** method in the Font class, to understand how to use the RGB colours.

Sinkhole

A sinkhole is a stationary object, shown by `sinkhole.png`. A sinkhole has **damage points**, which determines how much damage it inflicts on the player when they collide. A sinkhole has **30** damage points. The sinkhole should prevent the player from moving through it. If the player falls into (i.e. **collides**) with a sinkhole, it will inflict damage on the player (according to its damage points value). After collision, the sinkhole will disappear from the screen and the player should be able to move through the area it was placed at before.

Figure 4: Sinkhole

Wall

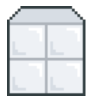


Figure 5: Wall

A wall is a stationary object, shown by `wall.png`. Like with the sinkhole, the player **shouldn't** be able to overlap with or move through the walls, i.e. the player must move around any areas on the level where blocks are being rendered. A wall has no damage points and cannot inflict damage on the player.

Log

Every time a sinkhole inflicts damage on the player, a sentence detailing this is printed on the **command line**, in the following format.

```
Sinkhole inflicts 30 damage points on Fae.  Fae's current health:  y/z
```

where `y` is the health points value after the damage was inflicted and `z` is the maximum health points value.

For example:

```
Sinkhole inflicts 30 damage points on Fae.  Fae's current health:  70/100

Sinkhole inflicts 30 damage points on Fae.  Fae's current health:  0/100
```

Your Code

You must submit a class called `ShadowDimension` 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
- Draw the player on screen
- Draw a block on screen
- Draw a sinkhole on screen
- Read the world file, and draw the corresponding objects on screen

- Implement movement logic for the player and her behaviour when colliding with a wall
- Implement the player's health points attribute logic and render the percentage on screen
- Implement the player's behaviour when colliding with a sinkhole
- 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 `ShadowDimension` class to help you get started, stored in the `src` folder. (2) All graphics and fonts 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).
  - `background0.png`: The image to represent the background.
  - `wall.png`: The image to represent a wall.
  - `faeLeft.png`: The image to represent the player when moving left.
  - `faeRight.png`: The image to represent the player when moving right.
  - `sinkhole.png`: The image to represent a sinkhole.
  - `frostbite.ttf`: The font to be used throughout this game.
  - `level0.csv`: The world file for the first level (there is only one world file for this Project).
- `src/` – The skeleton code for the game.
  - `ShadowDimension.java`: The skeleton code that contains entry point to the Game, a `readCSV()` method and an `update()` method that draws the background.
- `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, 01/09/2022 at 08:59pm**.

1. Clone the `[user-name]-project-1` folder from GitLab.
2. Download the `project-1-skeleton.zip` package from LMS, 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
    |
    ├─ ShadowDimension.java
    |
    └─ other Java files
```

On 09/09/2022 at 9:00pm, 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 09/09/2022 8:59pm 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 collision behaviour
- refactored code for cleaner design

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

- fesjakhbdjl
- yeah easy finished the player
- 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 be commenting as you go. *(Yes, we can tell).*
- 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.
- Any constant should be defined as a final static variable (**Note:** for Image and Font objects, use only final). 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 this form. 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 **8:59pm sharp**. Any submissions received past this time (from 9:00pm 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 this form. 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 **8** 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 – **5.5 marks**
  - Starting screen is implemented correctly: (**0.5 marks**)
  - The player's movement behaviour (images and bounds logic) is implemented correctly: (**1 mark**)
  - Wall and its' collision with the player logic is implemented correctly: (**1 mark**)
  - Sinkhole and its' collision with the player logic is implemented correctly: (**1 mark**)
  - The player's health points logic and percentage 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) – **2.5 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: (**0.5 marks**)
  - Code Style – visibility modifiers, consistent indentation, lack of magic numbers, commenting, etc. : (**1 mark**)