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

Project 1, 2017

Released: Friday August 25th
Due: Friday September 8th, 5pm

Overview

In this project, you will create a basic graphical game in the Java programming language. The graphics will be handled using the Slick library.

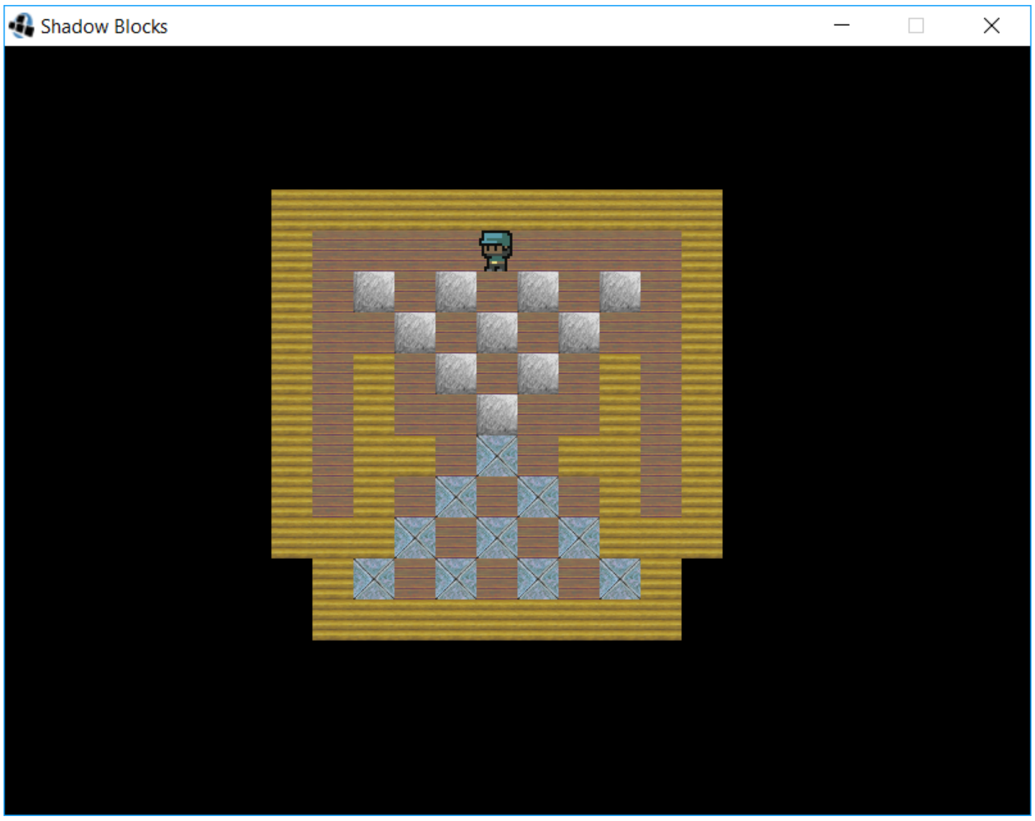
This is an **individual** project. You can discuss it with other students, but all submitted code must be your own work. You can use any platform and tools you like to develop the game, but we recommend using the Eclipse IDE, since that is what we are supporting in class.

In project 2, you will continue working on the game, and produce a complete playable product.

The purpose of this project is to:

- give you experience designing object-oriented software;
- introduce simple game programming concepts; and
- teach you to work with a simple object-oriented library.

Below is a screenshot of the game after having completed Project 1.



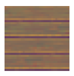
The game in its Project 1 state involves simply navigating a player character on a tile-based game board. The player can move through some tiles, but not through others.

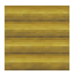
The player is controlled via the arrow keys. There is no goal at this stage – in project 2 the player will push stone blocks onto target squares to advance through the game.

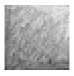
The rest of this document details exactly how the game is to work. You **must** implement all of the features described here. In project 2, you will have the opportunity to flex your creative muscles and extend the project beyond the specification if you so choose.

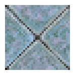
The game map

The world in this game is a two-dimensional grid of square tiles, each of which is 32 pixels wide and 32 pixels high. Remember that in Slick, the top-left corner is (0, 0). x therefore increases to the right, and y increases downwards. The tiles come in four types.

- **Floor:** 
  The floor tile is a plain background tile. The player can move freely through this tile.

- **Wall:** 
  The wall tile cannot be moved through.

- **Stone:** 
  For now, the stone tile is a plain background tile. The player can move freely through this tile.

- **Target:** 
  For now, the target tile is a plain background tile. The player can move freely through this tile.

- **Player:**
  While a bit different from other tiles, the player is treated as a tile, and should be loaded in the same way from `0.lvl`.

Each of the images, or "sprites", are provided to you in the *res* folder. You will need to load the world from a *level file.* One is provided to you in the *res* folder, called *0.lvl.* This file is in a comma-separated value (CSV) format. A sample file is provided below.

```
12,12
wall,0,0
floor,0,1
stone,0,2
target,0,3
player,0,2
```

The first line contains the width and height of the map. You will need this information so that the game world can be centred on the screen, rather than being drawn from the top-left corner.

Each successive line contains data in the format *type*, *tile_x*, *tile_y*. The type is either *wall*, *floor*, *stone*, *target*, or *player*. The coordinates are in tile counts; you will need to convert this to a pixel location on screen. Note that some tiles share a location. The earliest tile in `0.lvl` should be the first to be drawn, so that the latest tile is the one seen on screen.

The player

The player character is 32 pixels wide and 32 pixels high. Pressing (but not holding) the left, right, up, and down arrow keys moves the player one tile (32 pixels) in the appropriate direction, unless the player cannot move through the tile in that direction. To achieve this, you should make sure you store data on which tiles can and cannot be moved through in an appropriate data structure.

Your code

You have been provided with the skeleton code for four classes.

- **App**: this is the outer layer of the game, inheriting from Slick's `BasicGame` class. This class starts up the game, and handles delegation of `update` and `render` to the `World` class.
- **World**: this class represents the entire game world.
- **Sprite**: this class represents one sprite on the screen, including its graphics and its behaviour.
- **Loader**: this is a static class with functions to load the map from the CSV file.

The full code for `App` has been provided for you; the other classes have only method stubs. You can modify this code however you like – it is provided simply as a guide. Your implementation of the classes provided, as well as any that you add, should follow object oriented design principles, including abstraction, delegation, and polymorphism if appropriate.

Implementation tips

Here, we present some suggestions on how to go about doing the project. Feel free to ignore this advice if you think you have a better idea.

Loading the game map

We recommend using the `Loader` class to create the necessary sprites. An example on reading CSV files has been provided to you in past workshops; feel free to reuse this code with appropriate attribution.

Display

To draw the sprites, you can use the `Image.draw` method from Slick. This should be done in the `render` method of the relevant class(es). There is also a method to draw sprites centred; look at the Slick documentation available here.

Implementation checklist

This is a pretty big project, so to make it less scary, here's a list of the features you need to implement ordered in a logical way.

- drawing a sprite on the screen
- loading and parsing the level file
- displaying the map from the parsed data
- aligning the map correctly in the centre of the screen
- defining a class for the player
- movement of the player
- walls blocking the player

The supplied package

You will be given a package, `oosd-project1.zip`, which contains all of the graphics and data you will need, as well as some sample code to get you started. There is also a copyright notice detailing the source of the graphics; please follow the rules detailed at the provided links if you wish to use the graphics in other ways.

Submission requirements

The project must:

- be written in Java
- not depend on libraries other than the Java standard library and the Slick graphics library
- compile and run in Eclipse on the Windows machines in labs[1]
- contain no syntax errors (i.e. must compile fully on the command line)
- be submitted as a zipped Eclipse project containing **your student username** in the project name

Submission will take place on the LMS.

Coding style and best practices

Good coding style is a fairly subjective matter; often it depends on the company or team you're working with. For the purposes of the project we want you to think about:

- commenting **as you go** rather than after the fact. You *will* forget why you did something that weird hacky way after having not looked at your code for a week.
- proper use of visibility modifiers – unless you have a really good reason, member variables should be `private`. This is important because it helps to separate the interface from the implementation, ultimately leading to fewer logic errors.
- declaring constants appropriately. Constants should be marked `final`, and should be defined using CAPITAL LETTERS. Avoid magic numbers and strings in the interests of extensibility.
- future-proofing your code. This project will form the base of your project 2, so you should think about designing your project so that adding features will be as easy as possible. Obviously this is hard when you don't know future requirements... but this is software engineering, and requirements are typically not known until two weeks after the software is due.
- delegation. You should make sure that each class has a single well-defined purpose, and that every class handles its role fully. Large classes lead only to headaches, much like how large tutorials give your head tutor a headache.

Late submission

There is a penalty of **one mark** per day for late submissions, unless you have emailed appropriate documentation to the head tutor, Eleanor McMurtry, at `mcmurtrye@unimelb.edu.au`. If you submit late, you **must** email Eleanor with your username and the file you want us to mark; otherwise, we will simply mark the latest project submitted before the deadline. Note that the first mark is deducted at 5:01pm on Friday September 8th. This will not be negotiable.

[1] This is just so that there's a standard environment that everybody can access, in the interests of fairness.

Marking scheme

Project 1 is worth **8** marks out of a total 100 for the subject.

- Features implemented correctly: 4 marks
  - Level correctly loaded and displayed: 2 marks
  - Player displayed and able to be moved: 1 mark
  - Player correctly blocked by walls: 1 mark
- Coding style, documentation, and use of object-oriented principles: 4 marks
  - Avoiding magic numbers and strings: 0.5 marks
  - Encapsulation (data contained within appropriate classes): 1 mark
  - Delegation (functional decomposition and appropriate use of different classes): 1.5 marks
  - Use of visibility modifiers: 0.5 marks
  - Use of commenting: 0.5 marks