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

Project 2, 2017
Updated September 17th

Released: Saturday September 9th
Project 2A Due: Friday September 22nd, 11:59pm
Project 2B Due: Friday October 13th, 11:59pm

Overview

In this project, you will create a graphical puzzle game in the Java programming language, continuing from your work in Project 1[1]. 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.

You will not be required to design any aspect of the game itself; this document should provide all necessary information about how the game works. You will, however, be required to design the classes for your software solution before you implement it.

There are two parts to this project, with different submission dates.

The first task, **Project 2A**, requires that you produce a class design demonstrating how you plan to implement the game. This should be submitted in the form of a UML diagram showing all the classes you plan to implement, the relationships (e.g. inheritance and associations) between them, and their attributes, as well as their primary public methods. (Methods such as getters and setters need not be explicitly included.) If you so choose, you may show the relationships separately to the class members, but you **must** use correct UML notation.

The second task, **Project 2B**, is to complete the implementation of the game as described in the rest of this specification. You **do not** have to follow your class design; it is only there to encourage you to think about object-oriented principles before you start programming. If you do end up changing your design, we ask that you produce an **updated** UML diagram and submit it with your project.

[1] We will be providing a full working solution for Project 1 on Friday September 15th to allow for late submissions. You are welcome to use all or part of it, with appropriate attribution.

Shadow Blocks

Game overview

Shadow Blocks is a puzzle game where the player must solve block-based puzzles to advance deeper into the caves, while avoiding monsters that guard its depths. There are six *levels* in the game, each introducing new elements for the player to interact with. Broadly speaking, the game is divided into *blocks* that the player can interact with by pushing, *tiles* that have no active component yet add rules to the game, and *units* such as the player or the monsters that can wander each level.

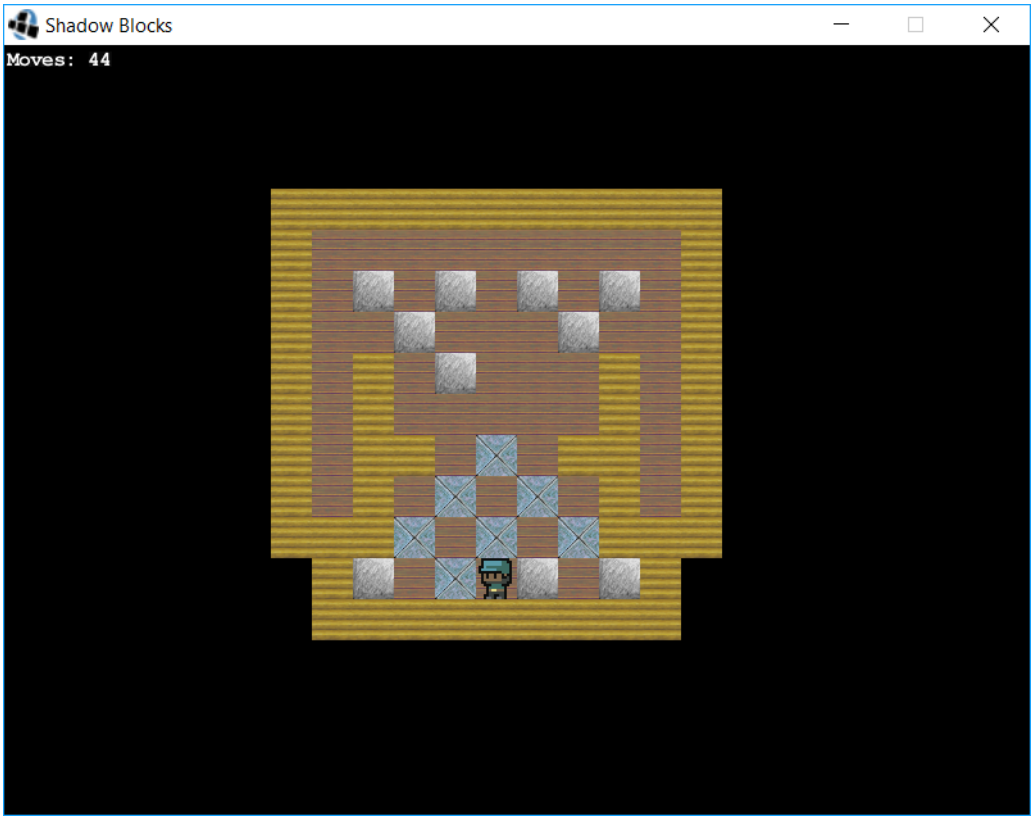
Gameplay

This is a real-time game; objects may move or change state even without the player pressing any keys. The game takes place in *frames*, at least sixty per second. On each frame:

1. All game objects have a chance to move. Units can move freely, whereas blocks can only move in a way dictated by their interaction with units.
2. If all blocks have reached their targets, the level is over and the game loads the next level.
3. The entire screen is "rendered", so that all previous drawing is cleared and the display reflects the current state of the game.

The player also has the opportunity to restart the current level by pressing R, and to undo recent moves by pressing Z. When the player undoes a move, all blocks should move to the position they were before the player last moved. The player should then return to the position they were before they last moved.

In addition, the number of moves the player has made during this level should be recorded and displayed in the top left corner. This counter should decrease when the player undoes a move.



Levels

There are six level files, each following the same specification as in Project 1. They are named `0.lvl`, `1.lvl`, and so on. Their names indicate the order that they should be loaded; the game should start

having loaded `0.lvl`. You will need to decide on an appropriate way to load these level files with the understanding that

1. each level file may be of different length; and
2. multiple sprites may exist in the same location.

Please note that your code will **not** be tested on any level files other than those with which we have provided you. Once a player has completed a level, the next level should immediately be loaded. You will not need to handle the final level, as the final level does not include any targets or blocks and therefore cannot be completed.

Units

There are four types of units in this game. For the purposes of this section, if the player attempts to move into a wall, this still counts as a move.

- **The player**: 
  The player has the same behaviour as in Project 1, except now the player can *push* blocks. The specific behaviour of this will depend on the block being pushed.
- **The skeleton**: 
  The skeleton is a mindless being that moves up once per second until it reaches a blocked tile, at which point it reverses direction and moves down until it reaches a blocked tile, and so on. If the player makes contact with the skeleton, the current level restarts.
- **The rogue**: 
  The rogue roams the caves, taking every opportunity they can get to steal from unsuspecting travellers. The rogue takes one step left each time the player moves, unless the rogue would walk into a wall; the rogue then reverses direction and moves right until they reach a wall, and so on. The rogue pushes any blocks they make contact with. If the player makes contact with the rogue, the current level restarts.
- **The mage**: 
  The mage likes to think that she is slightly cleverer than the rogue. She attempts to track down the player using Algorithm 1, as described below. This algorithm should run each time the player makes a move. If the player makes contact with the mage, the current level restarts.

Where $TILE\_SIZE = 32$:

Algorithm 1 Mage's movement

1: Let $dist_x$ and $dist_y$ be the x and y distances respectively between the mage and the player in pixels.
2: Let $\text{sgn}(x)$ be -1 if $x < 0$, and 1 otherwise.
3: If $|dist_x| > |dist_y|$ and the movement is possible, move $TILE\_SIZE * \text{sgn}(dist_x)$ pixels in the x direction.
4: Otherwise, if the movement is possible, move $TILE\_SIZE * \text{sgn}(dist_y)$ pixels in the y direction.

Blocks

There are three types of pushable blocks in this game.

- **Stone**: 
  The stone block is a simple block. When the player or the rogue moves into a tile occupied by

the stone, the stone should move one tile in the same direction as the unit which moved into it. If the stone cannot move in that direction due to a wall, neither the stone nor the unit should move.

- **Ice**: 
  The ice block behaves similarly to the stone block, except that when it is pushed it should continue moving in the direction it has been moved at a rate of one tile every 0.25 seconds. **Clarification**: The ice block should stop moving on contact with other blocks. The ice block has no particular effect on contact with a unit. If the ice block is moving and is pushed by a unit, it should change direction to the direction it has been pushed in.

- **TNT**: 
  The TNT block behaves identically to the stone block, except that if it is pushed into a Cracked Wall tile, both the wall and the block should be destroyed, and an Explosion effect should be created at the TNT block's position. **Note**: once the TNT block has been destroyed, its movement can no longer be undone.

Effects

There is one type of effect in this game; these are sprites that only exist for visual purposes, and do not interact with the game. The **explosion** effect is a sprite that should be displayed for 0.4 seconds, then be destroyed.

Tiles

There are six types of tiles in this game.

- **Floor**: 
  The floor tile has no special effects. Units can move freely through this tile.

- **Wall**: 
  The wall tile has no special effects. Units cannot move through this tile.

- **Target**: 
  The target tile must be covered by a block for the level to be completed. Units can move freely through this tile.

- **Cracked Wall**: 
  The cracked wall tile is identical to the wall tile, except that it can be destroyed by pushing a TNT block into it. **Note**: in the level file, the cracked wall tile is referred to as `cracked`.

- **Switch**: 
  The switch tile has the special effect that when a block is pushed onto it, it "unlocks" the door tile. There will only be one switch tile and one door tile in any given level. Once the block is pushed off the switch tile, the door tile is locked again.

- **Door**: 
  The door tile is controlled by the switch tile. When it is displayed, it is "locked", and units cannot move through it. When the door tile is unlocked by the switch tile, it should not be visible, and units can move freely through the tile.

Ending the level

When all target tiles that exist are covered by either stone or ice blocks, the current level is complete, and the next level should be loaded.

Implementation checklist

This project might seem a little worryingly large. Since there are a lot of things you need to implement, we have provided a checklist, roughly in the order that we recommend implementing them. Each feature comes with a number of marks that it is worth.

- All blocks are loaded and are able to be pushed by the player (1)
- Level completes when all targets are covered (0.5)
- Next level is loaded when the level is complete (0.5)
- Move count is tracked (0.5)
- Moves can be undone (1)
- Ice block slides when pushed (0.5)
- TNT block destroys cracked wall (0.5)
- Explosion effect is displayed and disappears (0.5)
- Enemy units are all loaded and displayed (0.5)
- Level is restarted on contact with an enemy (0.5)
- Rogue and skeleton move as described in their entries (0.5)
- Rogue can push blocks (0.5)
- Mage behaves according to Algorithm 1 or a similar approach to following the player (1)
- The switch tile hides the door tile when covered (1)

Customisation

Optional: one of the purposes of this project is to encourage creativity. We have tried to detail every aspect of the game design here, but if you wish you may customise **any** part of the game, including the levels, graphics, game rules, blocks etc. You may also add any features you want. However, you **must** implement and demonstrate all of the features in the above list to receive full marks.

For those of you with far too much time on your hands, we will hold an informal competition for the best game extension or modification. This will be judged by the lecturer and tutors. The top three will be demonstrated at the final lecture, and there will be a prize for our favourite. If you would like to enter this competition, please email the head tutor, Eleanor McMurtry, at `mcmurtrye@unimelb.edu.au` with a short description of the modifications you have made to the project. We can't wait to see what you come up with!

The supplied package

You will be given a package, `oosd-project2.zip`, which contains all of the graphics and data you will need. 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.

Reflection

For this project, we would like you to submit a short (500-1000 words) reflection on your experiences during this project. This should be in PDF format, and should include:

- an overview of any changes you made to your class design after submitting Project 2A, as well as your reasoning for these changes
- any difficulties you had during this project
- one key piece of knowledge you learned completing this project
- anything you would do differently if you did a similar project

Submission requirements

The project must:

- be written in Java
- not depend on any libraries other than the Java standard libraries `java.*` and the Slick libraries we have provided
- compile and run in Eclipse on the Windows machines in labs[2]
- 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, as well as a copy of your reflection in PDF format

Submission will take place on the LMS.

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

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.
- 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 12:00am on Saturday September 23rd for Project 2A, and at 12:00am on Saturday October 14th for Project 2B. This will not be negotiable.

Marking scheme

Project 2 is worth **22** marks out of a total 100 for the subject.

Project 2A is worth **6** marks.

Project 2B is worth **16** marks:

- Features implemented correctly: 9 marks, as detailed above
- 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 and sensible conventions: 0.5 marks
- Reflection: 2 marks (a thoughtful attempt is all that is required)
- Javadoc comments: 1 mark

Acknowledgement

This game was designed by Eleanor McMurtry.