The University of Melbourne
Department of Computer Science and Software Engineering
SWEN20003 Object Oriented Software Development

Project 2, 2015

Given: Friday 11th of September
Project 2a Due: Friday 25th of September, 5:00 PM
Project 2b Due: Friday 16th of October, 5:00 PM
Demonstration: Workshop, 19th–23rd of October

Overview

In this project, you will create a graphical kart racing game, *Shadow Kart*, in the Java programming language, continuing on from your work on Project 1.[1] You may use any platform and tools you wish to develop the game, but we recommend the Eclipse IDE for Java development. This is an individual project. You may discuss it with other students, but all of the implementation must be your own work. Like Project 1, you will be required to briefly demonstrate the game in a workshop.

You are not required to design any aspect of the game itself – this specification should provide all information about how the game works. However, you will be required to design the classes before you implement the game.

There are two parts to the project, with different submission dates:

The first task, **Project 2a**, requires that you produce a class design, using a diagramming tool of your choice, outlining how you plan to implement the game. This should consist of a UML class diagram, showing all of the classes you plan to implement, the relationships (inheritance and association) between them, and their attributes and primary public methods. If you like, you may show the class members in a separate diagram to the hierarchy diagram, but you must use UML notation. This must be submitted via LMS in pdf format.

The second task, **Project 2b**, is to complete the implementation of the game, as described in the manual which follows. You do not have to strictly follow your submitted class design if you feel that you now have a better design. Along with the code you are also expected to submit a reflection which discusses any difficulties you had with the project, at least one key piece of knowledge that you have taken away from this project and what you would do differently if you were to do this project again. If you had to make changes to your game design (Class diagram) in the process of implementation, state them in the reflection along with the reasons for these changes. You can also include a new Class diagram if you wish. The reflection should be no more than **1000** words in total.

The purpose of this project is to:

- Give you experience designing object-oriented software with simple class diagrams,
- Give you experience working with an object-oriented programming language (Java),
- Introduce simple game programming concepts (user interfaces, collisions, simple AI),

[1] Note: We will supply you with a complete working solution to Project 1; you may use part or all of it. It will be available on the LMS on Friday 18th of September, to allow for late submissions to Project 1.

Except when otherwise noted, you should assume that all of the rules from Project 1 still apply.

Shadow Kart

Game Manual

The animals on Farmer Catsplash's farm are running amok! The cow kicked his poor wife while she was milking her. The chickens have flown the coop. The pigs have made a dreadful mess, and the sheep have wandered off again. And the poor farmer's race karts are nowhere to be seen. This happens every year!

Sure enough, down at the race track, the animals are gathered to watch the big race. The farmer's favourite donkey revs his tractor. His dog pulls up in a kart made out of its own kennel. An elephant trumpets along and an octopus slithers out of the river and climbs into a kart.

You'll have to step on the pedal as hard as you can, and be ready to fight dirty if you want to win this year's barnyard race. It's tomato-throwing oil-spilling rocket-boosting mayhem down on the farm!

Game overview

Shadow Kart is a kart racing game where one player races against computer-controlled enemies to be the first one to the finish. The player controls Donkey, who is skilled at picking up various items found on the road to use or throw at his opponents. The goal of the game is to reach the finish line first, by any means necessary. Figure 1 shows the game in action.

In this game, the "world" is a two-dimensional grid of *tiles*. While the map is designed as a narrow track, the player is able to drive freely around the world (but of course, the player cannot drive on trees, mountains, water, and other types of terrain).

Gameplay

The game takes place in *frames*. Much like a movie, many frames occur per second, so the animation is smooth. Each frame:

1. Each millisecond: all four karts have a chance to accelerate forwards/backwards, rotate left/right, and/or use items. Enemy karts may accelerate or rotate automatically, depending on their racing strategy. The player moves, rotates, and/or uses items if the appropriate keys are being pressed. Then, each kart and on-screen projectile moves according to its current velocity and angle.

2. The "camera" is updated so that it is centered around the player's position.

3. The entire screen is "rendered," so the display on-screen reflects the new state of the world.

All of the karts accelerate forwards or backwards at a rate of 0.0005 px/ms, and rotate at a rate of 0.004 radians per millisecond, except in special circumstances explained later. Also, all karts are automatically decelerated by friction, which is dependent on the type of terrain underneath the kart.

Figure 1: A race, with the player character coming second to Dog

Controls

The game is controlled entirely using the keyboard. The **up** key causes the player to accelerate in the forward direction; the **down** key, in the backward direction. The **left** key rotates the player in the anti-clockwise direction; the **right** key, in the clockwise direction. If the player is carrying an item, either the left or right **control** key uses the item.

Karts and waypoints

There are four *karts* in the game, each with different skills and strategies. Figure 2 shows the starting position of each kart. All karts start at angle 0 radians, which should place all four karts on the starting line, facing north.

Kart	Start coordinates (x, y) in pixels
Elephant	1260, 13086
Donkey (player)	1332, 13086
Dog	1404, 13086
Octopus	1476, 13086

Figure 2: Starting coordinates for the four karts

(a) Elephant (b) Dog (c) Octopus

Figure 3: Enemy karts

The karts are divided into two general types:

- **The player** (Donkey). Controlled by you; able to pick up and use items.
- **Enemies**. The computer-controlled opponents, shown in Figure 3.

There are three enemy karts, determined to beat you in the race. Each has a different racing strategy. Enemy karts follow the same basic rules of movement and physics as the player, except that they cannot pick up or use items.

- **Elephant** – A ruthless fighter from the jungle, Elephant strives to win at all costs. He doesn't care about Donkey, and simply drives through the course as fast as he can.
- **Dog** – A man's best friend, Dog has developed a sense of honour. When she is ahead of Donkey (see *Ranking and game ending*), she accelerates at 0.9 times the normal rate ($0.00045px/ms^2$), to give you a chance to catch up. But, when she is behind Donkey, she accelerates at 1.1 times the normal rate ($0.00055px/ms^2$), to catch up to you!
- **Octopus** – Nobody knows much about Octopus, but he seems like kind of a jerk. When he's between 100 and 250 pixels away from Donkey, he drives straight towards the player! Otherwise, he follows the normal course. Is he even trying to win, or just get in your way?

Enemy karts are always accelerating forwards; they never stop accelerating. The direction of rotation, and the amount to rotate each frame, is determined by the kart's current destination.

The enemy karts know how to drive from the start to the end of the race track, by following a series of carefully placed *waypoints*. A waypoint is simply a pixel coordinate on the map. Each computer-controlled kart remembers which waypoint they are planning to visit next (for Octopus, even when following the player, he still remembers the next waypoint). All karts begin by planning to visit the first waypoint. Once a kart is within 250 pixels of the destination waypoint, they forget about that waypoint, and instead plan to visit the next waypoint in the sequence (so karts never actually reach the waypoint – they only get "close enough").

See the data file `data/waypoints.txt` for the exact location of each waypoint. See *Implementation tips* for more details on the waypoint algorithm.

Each frame, the kart rotates as far as possible to point towards the destination point. For Elephant and Dog, the destination point is always the next waypoint. For Octopus, this is sometimes the next waypoint, and sometimes the centre of the player's kart. The kart rotates 0.004 radians per millisecond, or less (if already facing the destination).

Collisions

An inevitable consequence of fast cars is fast crashes. A collision can occur between any two karts. When a kart hits another one, it loses all of its momentum, but is otherwise unharmed. Karts cannot be damaged or destroyed. The kart that got hit doesn't lose any momentum.

When a kart A is about to move, the game checks to see if the kart is about to move within 40 pixels of another kart, B, and if so, a *collision* occurs. Kart A does not move, and instead has its velocity set to zero.[2] There are no consequences for Kart B (although if B is driving towards A, it will likely collide with A soon enough – head-on collisions will be automatically handled this way).

Items and projectiles

(a) Oil can (b) Tomato (c) Boost

Figure 4: Items

There are a handful of special *items* (or "power-ups") scattered throughout the track, as shown in Figure 4. Like karts, each item has an (x, y) coordinate in pixels. Items can be used to slow down enemies or speed up the player, so you stand a better chance of winning the race. Only Donkey (the player) can pick up items.

At any point in time, the player may or may not be holding an item. The player's current item, if any, is shown in the lower right-hand corner (see *The Status Panel*). The player can pick up any item by moving within 40 pixels of its position. When picked up, an item will be removed from the world, and be set as the player's current item. If the player was already carrying an item, the old item will be deleted. The player can *use* their current item by tapping the **control** key. Using an item deletes it from the player's hand, and has some effect, depending on the item type.

- **Oil can** – Creates an *oil slick* behind the player. The oil slick is placed 40 pixels away from the player, in the opposite direction to that which the player is facing. This is useful for delaying enemies that are coming up behind you.
- **Tomato** – Can be thrown at enemies in front of the player. A *tomato projectile* is created 40 pixels away from the player, in the direction the player is facing. The tomato projectile is given a fixed velocity of $1.7px/ms$ in the direction the player is facing. This is useful for delaying enemies that are just in front of you.
- **Boost** – Activates a rocket afterburner that increases the player's acceleration for a short time. The boost lasts for 3 seconds, and during that time, the player's acceleration is fixed at $0.0008px/ms^2$ in the forwards direction (regardless of whether or not the player is pressing the **up** or **down** keys). This is useful for catching up to enemies that are far ahead.

[2] You may recall that this is the exact same algorithm for dealing with crashing into blocking tiles in Project 1.

See the data file `data/items.txt` for details on the locations of the items.

The *oil slick* and *tomato projectile* objects are dangerous to all players. The only difference (besides appearance) is that oil slicks are stationary, while tomato projectiles move with a fixed velocity. Unlike karts, tomato projectiles are not subject to friction. However, if a tomato projectile hits an undrivable tile (such as mountain or water), it is immediately destroyed. If a kart hits an oil slick or tomato projectile, the slick/tomato is destroyed, and the kart is sent into a "spin" for 0.7 seconds. When a kart is spinning, its driver has no control over the kart: it always accelerates forward (at the normal acceleration rate), and rotates clockwise at a rate of 0.008 radians per millisecond. Therefore, hitting an oil slick or being hit by a tomato projectile can seriously throw a kart off course, costing valuable seconds.

There can be any number of oil slicks or tomato projectiles in play at a time.

The status panel

Figure 5: The status panel

The overlay in the bottom-right corner of the screen (pictured in Figure 5) is known as the *status panel*.[3] It shows the player's current status:

- Ranking: The lower bubble shows the player's ranking (see *Ranking and game ending*).
- Item: The upper bubble shows the item the player is currently holding, if any.

Ranking and game ending

The four karts are continuously ranked in order from first to fourth, so it is always possible to tell who is winning at any given time. This information is used for three things: to show the player their current ranking (see *The Status Panel*), to control the racing strategy of Dog, and to determine the eventual winner.

Each kart is *ranked* either 1st, 2nd, 3rd or 4th. This race position is *just an approximation* – karts are ranked according to their y coordinate, so whichever kart is furthest north counts as 1st.

The game ends when the player's y coordinate is less than 1026 (the player has crossed the finish line). Once the game is over, the player can no longer control the kart (it gradually rolls to a halt), but the computer-controlled karts can still play. A message is displayed in the centre of the screen, telling the player their ranking: "You came 1st!" (or 2nd, 3rd or 4th).[4]

[3] The rendering of the status panel has already been done for you – you just need to call the appropriate method. See `Panel.java` in the supplied package. Note that you will need to write the code to calculate the player's ranking.

[4] Be careful that you don't let the player's ranking change once the game is over! Remember that the other karts might overtake the player afterwards – this should not change the rankings.

Implementation tips

Game loop

We have decided to change the way the `delta` value is used from Project 1. Rather than multiplying everything by `delta`, we are going to call `World.update` `delta` times per frame. We have supplied a new `Game.java`, which no longer passes `delta` to `World.update` – you can assume this will be called once per millisecond. It also passes a `use_item` argument, which is `true` if the user is pressing **control**. You should remove all references to `delta` in your code, and change your `World.update` to the following signature:

```
public void update(double rotate_dir, double move_dir, boolean use_item)
```

Waypoints

Figure 6 shows how computer-controlled karts navigate via waypoints. The small red circles represent Waypoints #7, #8 and #9, while the large blue circles represent the 250-pixel radius around each waypoint that karts consider to be "close enough." The solid red line shows the path that will be taken by the Elephant kart shown, assuming no interference by other karts.

Note the dashed red lines – these show the kart's intended goal of reaching the centre of the waypoint. When the kart hits the edge of the blue circle for Waypoint #7, it changes course and begins heading for Waypoint #8, and shortly after that, it hits the edge of the blue circle for Waypoint #8, changing course for Waypoint #9, and so on.

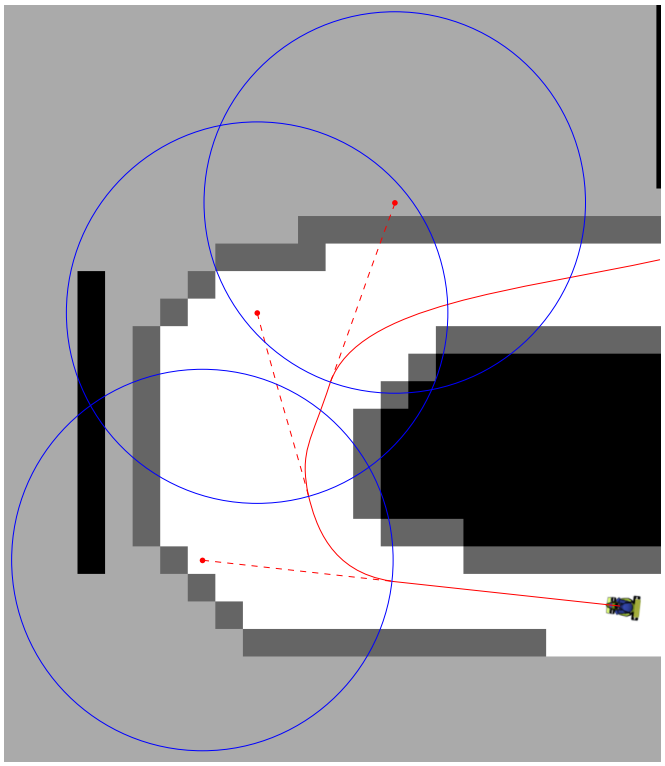

Figure 6: How waypoint navigation works

Implementation checklist

This project may seem daunting. As there are a lot of things you need to implement, we have provided a checklist, ordered roughly in the order we think you should implement them in. Next to each item is the number of marks that particular feature is worth, for a total of 9.

- All the kart and item types implemented and visible. (1)
- Display bottom status panel, with player's ranking and item. (0.5)
- Collisions between karts. (1)
- Enemy karts can follow the waypoints and complete the race. (1)
- Enemy karts implement their individual racing strategies. (1)
  - Dog should adjust her speed based on relative ranking to player.
  - Octopus should drive towards the player when nearby.
- Player can pick up and use items. (0.5)
- Boost properly adjusts the player's acceleration. (1)
- Oil slick and tomato projectiles are created and displayed. (0.5)
- Tomato projectiles move properly, and are destroyed if they hit a blocking tile. (1)
- Karts collide properly with a tomato projectile or oil slick. (1)
  - Tomato/slick is destroyed and karts sent into a spin.
  - Neither the player nor enemy karts can control the kart while spinning.
  - The spin lasts for the correct length of time.
- Game over message, showing the player their final ranking. (0.5)

Customisation

Optional: The purpose of this project is to encourage creativity. While we have tried to provide every detail of the game design, *if* you wish, you may customise *any* part of the game (including the graphics, karts and items, map layout, waypoints and game rules). You may also add any feature you can think of. However, you ***must*** implement all of the features in the above checklist to be eligible for full marks.

To encourage students with too much free time, we will hold a competition for "best game extension or modification". It will be judged by the lecturer and tutors, based on the demonstrations, and three prizes will be offered. We look forward to seeing what you come up with!

The supplied package

You will be given a package, `oosd-project2-package.zip`, which contains all of the graphics and other files you need to build the game. You can use these in any way you want. Here is a brief summary of its contents:

- `assets/` – The game graphics and map files.
  - `map.tmx` – The map file. You should pass this as input to Slick's `TiledMap` constructor.
  - `tileset.tsx` – The tileset file; automatically included by `map.tmx`.
  - `tiles.png` – The terrain graphics; automatically included by `tileset.tsx`.
  - `panel.png` – An image file you may use for the panel at the bottom-right of the game.
  - `karts/` – Contains all of the kart image files.
  - `items/` – Contains all of the item image files, as well as the oil slick and tomato projectile images.
- `data/` – Text files with waypoint and item locations. These files contain tables with full explanations.
  - `waypoints.txt` – Positions of all the waypoints.
  - `items.txt` – Positions of all the items.
- `src/` – Various source files supplied for you.
  - `Game.java` – Updated version which passes `use_item` and does not pass `delta`.
  - `Panel.java` – A small class for rendering the status panel. (You don't have to use this.)

Legal notice

The graphics included with the package (`tiles.png`, `karts/*`, `items/*`) are primarily derived from copyrighted works from the games *SuperTuxKart* (licensed under Creative Commons Attribution-ShareAlike 3.0) and *The Battle for Wesnoth* (licensed under the GNU General Public License, version 2 or later). You may redistribute them, but only if you agree to the license terms.

For more information, see `assets/README.txt`, included in the supplied package.

Technical requirements

- The program must be written in the Java programming language.
- The program must not depend upon any libraries other than the Java standard library, and the Slick graphics library we have provided.
- The program must compile and run in Eclipse on the Windows machines in the labs. (This is a practical requirement, as you will be giving a demo in this room; you may develop the program with any tools you wish, as long as you make sure it runs in this environment.)

Submission

Submission will be through the LMS.

Class diagrams for Project 2a must be submitted in pdf format, via LMS, by the deadline for Project 2a.

For project 2b, you will need to create a zip file of your eclipse project, along with your reflection in pdf format. These must be submitted via LMS by the deadline for Project 2b.

We will publish submission links closer to the submission date through the LMS.

Demonstration

You will be required to demonstrate the game engine for your tutor in your workshop in the week of the 19th–23rd of October. Your tutor will check out the submitted version of your code (i.e., the checked-in version of the code as of the deadline), then ask you to the front of the room for a few minutes. You will be asked to demonstrate various features of the engine working correctly. Not attending the demonstration will incur a 1 mark penalty.

Late submissions

There is a penalty of 1 mark per day for late submissions without a medical certificate. Late submissions will be handled by Mat Blair, the head tutor for the subject. If you submit late, you *must* email Mat Blair at `mathew.blair@unimelb.edu.au` with your login name and the zip file you wish for us to consider when marking (if you don't, we will simply mark the latest version you have submitted by the deadline).

Marks

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

- Project 2a is worth **6** marks.
- Project 2b is worth **16** marks.
  - Features implemented correctly – **9 marks** (see *Implementation checklist* for details)
  - Code (coding style, documentation, good object-oriented principles) – **4 marks**
  - Reflection - **2 marks**
  - Documentation (javadoc) – **1 mark**

Acknowledgement

This game was designed by Matt Giuca.