Robot Island

COMPGC01 Coursework

By Geraint Ballinger and Lucas Valtl

14th of December 2016

Requirements List

Overview

Robot Island is a java application that applies the requirements of the COMPGC01 coursework to a game like environment. The main goal of the game is to move a robot along a racetrack and achieve the fastest time possible without running out of battery. The application is provided as an eclipse folder as well as an executable .jar file.

Fulfilled Coursework Requirements

- **JavaFX GUI:** The application should be implemented as a JavaFX GUI application using buttons, listeners, etc.
    - Implemented using JavaFX using buttons, button listeners, labels, key listeners, images and rectangles.
- **Robot Representation:** The robot is represented by a rectangle, or an image
    - The robot is represented by a rectangle. The rectangle, however, is filled with an image. The image changes depending on the state of the robot (see animation below).
- **Robot Movement:** The robot can a) move forward or backward with a given speed b) rotate left/right with a given angle.
    - The robot can move (accelerate) forwards and backwards. It also turns left and right when not moving forward. In addition, it can move forward in combination with moving left or right.
- **Robot Control:** A user can control the robot through a set of GUI commands such as buttons and text fields, etc, or keyboard keys.
    - The arrow keys are used to control the robot: the up key causes forward motion whereas the down key causes backward motion, the left key rotates the robot anti-clockwise while the right key rotates the robot clockwise.
- **Maze:** The robot can move in a maze that can be represented by a set of connected lines.
    - The robot moves through a track (similar to a maze, however the beginning and end points are the same). The track is represented by rectangular blocks drawn on the screen, which are invisible. However, a direct representation of the rectangle positions is given via a background image, allowing for a more flexible design.
- **Collision detection / Alert on Collision:** A text or audio alert is triggered if a move would cause the robot to collide with one of the walls (lines representing a wall).
    - Collisions are recognised using a three phased collision detection algorithm in order to be both accurate and efficient:

        - Broad phase collision detection: Collision detection is only performed on rectangles nearest to the robot. Blocks that surround the robot are passed to the 1st narrow phase for further collision checking.
        - 1st narrow phase collision detection: JavaFX's built-in bounding box collision detection is used to check for a collision. If this phase recognises a collision, the 2nd narrow phase is triggered.
        - 2nd narrow phase collision detection: Separating Axis Theorem based collision detection is used to precisely detect a collision. If this phase recognises a collision, an alert is triggered (see below). Almas Baimagambetov's tutorial "GameDev: Collision Detection - SAT" [1] was used to implement the separating axis theorem into the application.
    - If the 2nd narrow phase of the collision detection algorithm detects a collision, an audio alert is triggered. In addition, a visual alert is also triggered by changing the robot's eyes to crosses.
- **Battery usage:** The robot has an initial battery charge that decreases each time the robot moves one unit of distance. The robot stops if battery charge is equal to zero.
    - Battery decreases with each unit of distance travelled. The faster the robot moves (the more distance covered per second), the more the battery charge will decrease. When the battery level reaches zero, the robot stops moving.
- **Battery alert:** A text or audio alert is triggered when the robot charge is less than 10% of the maximal charge.
    - When the robot's charge level reaches 10%, a continuous audio alert is triggered in addition to a low battery symbol being displayed on the robot's face. A text label is also updated to inform that the battery is low.
- **Live Updates:** A text (java text area for example) is updated live with the coordinates of the robot during movement; it's angle, distance crossed and battery level.
    - A developer pane on the screen is updated with live coordinates (x and y) of the robot center position, it's angle (direction the robot's face is pointing), the distance travelled and the battery level.
- **Logging:** Record all commands executed by the robot in a text file with a timestamp.
    - A logger is used to record all commands executed by the robot, including a timestamp in the following format:
        - Dec 14, 2016 12:31:06 PM robot.Movement decelerate INFO: decelerate
- **Execute from text file:** Read commands from a text file and execute them by the robot at once.
    - After the player selects an appropriate text file using a file chooser, the robot executes all commands in that text file. The text file is validated for illegal commands before the commands are executed.

- **Differential Steering:** Study differential steering of wheeled robots and provide an implementation.
    - User instructions are translated from a 'unicycle model' of robot movement to the ''differential model': users can press the left or right arrow keys to apply torque to the robot, and can press the up or down keys to apply a force upwards or downwards. In other words, the user controls the angular velocity, ω and speed of the robot (at the centre), ν:

$$\frac{dx}{dt} = \nu\cos(\phi)$$
$$\frac{dy}{dt} = \nu\sin(\phi)$$
$$\frac{d\phi}{dt} = \omega$$

      where x is the robot's x coordinate, y is the robot's y coordinate, v is the robot's speed, and ϕis the robot's orientation.
    - In the differential model, however, as detailed by G.W. Lucas [2], the new coordinates and orientation depend on both wheel velocities:

$$\frac{dx}{dt} = \frac{R}{2}(v_r + v_l)\cos(\phi)$$
$$\frac{dy}{dt} = \frac{R}{2}(v_r + v_l)\sin(\phi)$$
$$\frac{d\phi}{dt} = \frac{R}{L}(v_r - v_l)$$

      where x is the robot's x coordinate, y is the robot's y coordinate, ϕ is the robot's orientation, R is the radius of the wheels, L is the axle length between the wheels, and v_l and v_r are the speeds of the left and right wheels.
    - As explained in the Kansas State Study Guide [3], the velocities of each wheel may be calculated from the overall angular velocity and speed of the robot (at the centre) by the following equations:

$$v_r = \frac{2v + \omega L}{2R}$$
$$v_l = \frac{2v - \omega L}{2R}$$

where v is the robot's overall speed, $\omega$is the robot's angular velocity, R is the radius of the wheels, L is the axle length between the wheels, and v_l and v_r are the speeds of the left and right wheels.

    - In the current implementation, a difference in speed triggers an animation (see below).
- **Load from XML file:** The parameters corresponding to the robot geometry such as the dimension of the robot, the distance between the two wheels, the wheels radius, etc are stored in an XML file and loaded when the application starts. The implementation provided in requirement (12) should be adapted accordingly.
    - An XML file is used to initialise the robot. A user may change the type of robot (currently either 'fast' or 'slow') in the source files, and the corresponding parameters from this type of robot will be read from the XML file and loaded into the robot's variables.

Additional Features

- **Game like dynamics:** The robot simulator is turned into a game by introducing game like features that allow the player to measure their performance. Performance is measured as a time per lap, wherein a lower time equates to a higher score, thereby motivating the player to reach a higher score and continue playing.
    - A splash screen shows the game's name and initial instructions
    - A game over screen is displayed when the battery runs out and gives a hint on how to improve.
- **Sound effects:** Dynamic sounds are played depending on the robot's state and its interaction with the environment
    - Background music is played continuously
    - A sound is played when the player crosses the finish line
    - A sound is played when the player achieves a new high score
- **Lap high scores:** If a player achieves a new high score, the new high score is recorded and the player receives feedback. The high score is loaded when the game is opened again.
    - Highscore feedback is given visually on the robot's face as well as in audio form
    - The score is saved to a text file in order to be accessed when the game is used next.
    - The high score is always displayed below the current and last lap time.
- **Cheating-prevention:** To avoid the possibility of gaining an unfair advantage, and increase the reliability of the scoring system, an anti-cheating system is used.
    - The high score file is encrypted every time a new high score is written and decrypted when loading the high score into the game. The player can never

change their high score by manually changing the text file. Encryption was implemented with the help of a tutorial provided on codejava.net [4].
    - To deter players trying to increase their score without performing a full lap, a checkpoint (pseudo-finish line) is implemented on the opposite side of the map. Robots that have not driven over the checkpoint are not able to record a new lap, therefore forcing potential cheaters to do a full lap.
- **Animation:** Animation is provided to better inform the user of their actions and the state of the robot.
    - Animation is implemented by loading different images into a nested list. A different image is called from the nested list depending on the robot's current state, and loaded as an image fill into the robot. When the robot is in a normal state, an image picture is loaded depending on the speed and direction of the robot. However, when the user needs to be informed about a special situation (e.g.low battery), a more appropriate picture is displayed.
    - The "eyes" are used to inform the user of the direction and speed of the robot. A difference in wheel speeds will be indicated by a corresponding eye animation.
    - A low battery symbol indicates a low charge level.
    - Two exclamation marks indicates a new high score.
- **3D like 2D graphics:** The user is given the impression that the robot is in a 3D environment
    - A 2D background picture with 3D like elements (mountains, signs) is used to immerse the user into a world where a robot is on an island and driving on a track around a mountain. A 3D appearance is created using shadows (e.g. in the mountain) and diverging angles (e.g. on the sign showing the lap times).
    - In addition, the current lap time text display is presented in faux 3D using a perspective transformation function.
    - Buttons were styled using a CSS stylesheet using different background layers that were offset to create a 3D look. The tutorial from Jasper Potts [5] was used as a reference for this.
- **Time trial mode:** The user can race a "ghost" robot which replicates the user's high score lap
    - When time trial mode is switched on, a faded out robot will appear each time the user passes the finish line. This new robot will simulate the user's high score lap, providing visual feedback on the quality of their current attempt and hopefully increase competitiveness.

Copyright Details

- Soundtrack: Chibi ninja by Eric Skiff is licensed under CC BY 4.0.
- Other sound effects: Shut_Down1, Space_Alert3, Emerge4, Emerge9, Sweep1, Power_Up3 by Morten are licensed under CC BY 1.0.
- Font used in Game Logo: "pixelated" by "Greenma201" is licensed under CC BY SA 3.0.

References

[1] Almas Baimagambetov, "Game Dev: Collision Detection – SAT." 2016. [Online]. Available: https://www.youtube.com/watch?v=Sv42pfgiAI4. [Accessed: 14-Dec-2016].

[2] G. W. Lucas, "A Tutorial and Elementary Trajectory Model for The Differential Steering System." [Online]. Available: http://rossum.sourceforge.net/papers/DiffSteer/. [Accessed: 14-Dec-2016].

[3] "The Unicycle Model — LabVIEW Robotics Programming Study Guide." [Online]. Available: http://faculty.salina.k-state.edu/tim/robotics_sg/Control/kinematics/unicycle.html. [Accessed: 14-Dec-2016].

[4] "File Encryption and Decryption Simple Example." [Online]. Available: http://www.codejava.net/coding/file-encryption-and-decryption-simple-example. [Accessed: 14-Dec-2016].

[5] Jasper Potts, "Styling FX Buttons with CSS." [Online]. Available: http://fxexperience.com/2011/12/styling-fx-buttons-with-css [Accessed: 16-Dec-2016].