**Mini Project 2 – Breakout**

**Expected grade: A+**

**Running the game**

To run the game, run **run.bat** in the root directory of the project. If you dislike .bat files, you can run **java -cp "bin" breakout.Main** from a terminal in the root directory of the project. Alternatively, you can open the project in Eclipse and run it from there.

**Testing**

To test the program, I used the Debug class to output information at various parts of the program. Below is a table of some of the things I tested.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test** | **Description** | **Expected result** | **Result** |
| Side collisions | Setup a test game so the ball would collide with the side of a brick | Ball correctly bounces of the side of the brick (x velocity inverted) | Worked |
| Multiple lives | Let the ball collide with the floor when the score is less than 500 | Lives decrease until the all lives gone, at which point the game outputs “You lost!” message | Worked |
| Level transition | I setup a key bind “l” to move the game level forward | The game goes forward one level until it outputs “You won!” message | Worked |
| Multiple collisions destroy a breakable brick | Setup the game with only breakable bricks | The breakable brick transitions from normal to slightly broken to very broken to destroyed (disappears) | Worked |

**Features**

* **Multiple levels.**

The game contains multiple levels progressing in difficulty. Each level is stored in an XML file (see **Extensible map framework** below). When the number of destructible bricks (see **Multiple brick types**) in a level is zero, the game moves on to the next level until the final level is finished, at which point the user has won.

* **Ball swerve on bat hit.**

A swerve/spin is applied to the ball when it collides with the bat. This adds another element to the game and in the future it could be used to make more interesting levels. The amount of swerve is based on the bat speed, bat speed, the x-direction of the ball, and a constant.

* **Bat acceleration involving friction.**

The bat is accelerated when the user presses the level/right arrow keys. When the user releases the arrow keys, the bat keeps moving but slows down due to a friction constant.

* **Extensible map framework.**

This framework uses XML files to store level data and uses a character notation to define the brick layout. For example, “---bbbu---“ would be displayed as “no brick, no brick, no brick, breakable, breakable, breakable, standard, no brick, no brick, no brick”. Also stored in the XML files are level name, version, minimum ball speed, bat width, and bat speed transfer mod (ratio of bat speed given to ball).

* **Multiple lives**

When the ball hits the floor and the score is less than 500, a player loses a life and must restart the level (the score is reset to zero and lives are decremented). If the number of lives is zero, the game displays a message saying the user has lost.

* **Multiple brick types**

There are three types of brick in the game: **breakable**, **standard**, and **indestructible**. The breakable brick has three states: not broken, slightly broken, and very broken. When the very broken state is hit by the ball, the brick is destroyed (it disappears). The standard bricks only take one hit from the ball to be destroyed, and the indestructible bricks cannot be destroyed at all.

Each destructible brick adds to the user’s score when hit by the ball.

* **Side collisions**

The ball can collide with the side of a brick or the ball and act realistically (inverts the x direction of the ball). The ball also separates itself from the thing it collided with using a “separation vector”. The solution I used for this is defined in breakout.objects.Collision.

There are a few edge cases where the ball may collide with the corner of a brick/bat and rebound the wrong way. This is due to the overlaps of both dimensions (x and y) being roughly equal so the game doesn’t know whether the ball collided on the side of the brick or the top/bottom. This could have been fixed by checking the direction of the ball and its relative position to the brick, but I disliked this solution as it would deny extensibility in the future, such as the ability to rotate bricks.

The best solution would have been to use the separating axis theorem and calculate the separation vector that way, and use the normal of the side collided with to reflect the velocity vector of the ball. This would have been overkill for this basic game and would have taken a fair bit of time to get working (implementation and debugging). Given more time however, I would like to implement it.