CS308
GizmoBall
Group: TMurray2

John Lewis
Adam McGhie
Alan McPhee
Hristo Hristov
Kyle Robertson

Revised Specification----------1
Use Cases----------2
Class Diagram Description----------7
View Package----------7
Model Package----------8
Controller Package----------9
Gui Mockup----------10
Physics Loop----------12
Triggering System----------13
Project Plan----------15

Revised Specification

- The system should allow the user to be able to build the custom levels that the user can then play on the system.
- The user should be able to freely edit each level by moving or rotating triggers/ bumpers in a 90 degree clockwise direction in each level to however they choose. They can also add and remove bumpers/ triggers if and when they choose as long as they are not running the game mode.
- The game should be able to switch between build mode and running mode at any time.
- The user should be able to create the flippers and then be able to control them while playing the gizmoball game.
- The triggers should should rotate 90 degrees once a key is pressed. Once the key is released, the trigger should go back to its former position.
- Should also allow the user to save and load the user's custom game configurations to and from a file.
- The action of the bumpers and the triggers should be connected to an action such as a keyboard key or a mouse button to allow the user to interact with the system.
- Connect a particular gizmo's trigger to a particular gizmo's action.
- In building mode, you can also add a ball to the playing area.
- Each bumper should change colour when a collision is detected by the gizmo.
- The outer wall of the game window also acts as a bumper and the gizmoball should defect off of this and change trajectory in accordance to this.
- System should monitor gravity and friction so that the ball falls and rises realistically and smoothly within the system, much like a ball would in real life.
- The system should be actively monitoring the velocity of the ball to account for changes in friction and gravity.
- Quit the game in both running and building mode.
- Shoot the gizmoball out if it enters the absorber bar at the bottom of the game window
- Calculate the gizmoballs velocity using the coefficient of reflection of any surface that it collides with.

Use Cases

Add Gizmo
Preconditions: In edit mode.
Triggers: Add Gizmo button is pressed.
Basic course of events:

i. A new window pops up to prompt the user on the details of the gizmo.
ii. User chooses location to place gizmo

Alternative Paths:

i. If user places gizmo in an unsuitable location (such as already on another gizmo) then an error arises saying the Gizmo is unable to be added.

Postconditions: the gizmo is added to the game board.

Remove Gizmo
Preconditions: In edit mode and have gizmo's already present in the system.
Triggers: Delete Gizmo button is pressed
Basic course of events:

i. User selects the gizmo they wish to remove.
ii. Gizmo is removed from the system

Alternative Paths:

i. If user tries to delete a gizmo that isn't there, an error arises saying that there is no Gizmo to be removed.

Postconditions: the gizmo is removed from the game board.

Move Gizmo
Preconditions: System in edit mode, and there is a gizmo to move
Triggers: Move Gizmo button is pressed
Basic course of events:

i. User selects the Gizmo they want to move
ii. User selects the new location for the gizmo

Alternative paths:

i. If user tries to move a gizmo and place it in an unsuitable location, then show error saying that the gizmo can't be placed there.
ii. If there are no gizmo's present, show error saying that there are no gizmo's to be moved.

Postconditions: The gizmo is moved to the new location

Rotate Bumper/Trigger

Preconditions: System is in build mode with already having gizmos to rotate

Triggers: Rotate Gizmo button pressed

Basic Course of Events:

i. User selects the gizmo they wish to rotate
ii. User then clicks Rotate Button for the gizmo to be rotated 90º clockwise

Alternative Paths:

i. User adds a gizmo the level
ii. User then can select this gizmo for rotation
iii. User then clicks Rotate Button for the gizmo to be rotated 90º clockwise

Alternative Paths:

i. If user rotates gizmo which interferes with another gizmo, then show error stating that the gizmo cannot be rotated.

Alternative Paths:

i. If there are no gizmo's present in the system, show error stating that the user cannot rotate any gizmo's as there are no gizmo's present.

Save Game

Preconditions: Have a game configuration present for the user to save

Triggers: Save game button pressed

Basic Course of Events:

i. Prompt allows user to enter a name for the game they wish to save
ii. File is saved to a common directory

Alternative Paths:

i. User must save game in correct format for it to be loaded at a later stage

Postconditions: the game is saved and can be loaded at a later date

Load Game

Preconditions: Have a saved game file that the user can load

Triggers: Load Game button pressed

Basic Course of Events:

i. Prompt allows user to choose which save game file they wish to load from directory
ii. The game is then loaded into the system
iii. Saved game configuration can then be edited or played.

Alternative Paths:

i. If user selects a file that is unsupported by the system or if the file is corrupt for example, then the system should provide an error saying that the file is unsupported by the system and is unable to load it.

Postconditions: the game is loaded and can be edited and played or saved again.

Absorb Ball
Preconditions: Game is in play mode, gizmoball and absorber are present
Triggers: Gizmoball falls into the absorber block at the bottom of the game window
Basic Course of Events:

i. Gizmoball collides with the absorber block
ii. Gizmoball is instantly transported to the new coordinate at the bottom right hand corner of the game screen.

Alternative Paths:

i. If no absorber is present, then the ball should fall to the ground and the ball will cease to exist.

Postconditions: The gizmoball is then shot from the new position in an upward fashion.

ShootBall
Preconditions: Game is in play mode, gizmoball and absorber are present
Triggers: Absorb Ball has just been performed.
Basic Course of Events:

i. Velocity and direction of Gizmoball changed to 50L/sec in an upwards direction

Alternative Paths:
Postconditions: Gizmoball is shot up with it's new velocity

Connect Key Press
Preconditions: In build mode, with having gizmo's present.
Triggers: By pressing the command to set a Key to that particular gizmo
Basic course of events:

i. User selects gizmo that is already on the level window
ii. Window prompt shows up asking if user wants a keypress associated with that gizmo
iii. User selects key they wish to associate with gizmo

Alternative paths:

i. User adds a gizmo to the level window
ii. Window prompt shows up asking whether user wants to associate keypress with gizmo
iii. User selects key they wish to associate with gizmo

Alternative Paths:

i. If no gizmo is present, then if user want to connect a key press to a gizmo then show error stating that this cannot be done.

Postconditions: The gizmo has a keypress associated with the gizmo

Connect Gizmo-Action
Preconditions: In build mode, with having more than one gizmo present
Triggers: By the ball hitting that particular gizmo to change its behaviour
Basic Course of Events:

i. User selects gizmo that is already on the level window
ii. Window prompt shows up asking user wants to connect gizmo-action
iii. User selects what element of the bumper/ trigger they wish to change e.g. colour, movement

Alternative Paths:

i. User adds a gizmo to the level window
ii. Window prompt shows up asking whether user wants to connect gizmo action with bumper/ trigger
iii. User selects element of the bumper/ trigger they wish to change

Postconditions: The gizmo has an action connected to the gizmo in which the behaviour of the gizmo will be changed on this action.

Switch to Running Mode
Preconditions: In build mode
Triggers: By wanting to run recently edited level
Basic Course of Events:

i. User has finished editing level and wish to change back to Running Mode
ii. User presses Running Mode button to change back to the Game Mode.

Alternative Paths:

i. User should be unable to switch to running mode while already in running mode.

Postconditions: Game is returned back to Running Mode

Switch to Build Mode
Preconditions: In running mode
Triggers: By wanting to edit a level that currently exists
Basic Course of Events:

i. User has finished playing game and wish to make changes to the level they are playing
ii. User presses Build Mode button to change the game into the Edit Mode

Alternative Paths:

i. User should be unable to switch to build mode while already in build mode

Postconditions: Game is returned back to Build Mode

Add Ball to Playing Area

Preconditions: Game must be in Edit Mode

Triggers: By wanting to create a new location for the ball or velocity, click add ball button

Basic Course of Events:

i. Prompt pops up requesting user for velocity and position on screen for the ball to spawn from.
ii. Choose location and velocity of ball

Alternative Paths:

i. If location is inappropriate (on top of another gizmo (except an absorber) then produce an error message saying that the location of the ball is inappropriate
ii. If velocity is inappropriate (too large or too small a velocity) then produce an error message saying that the location of the ball is inappropriate

Activate Flipper

Preconditions: Flipper is present in the game, flipper is bound to a key and game is in run mode

Triggers: the user presses a key that is bound to a flipper

Basic Course of Events:

i. User presses key that flipper is bound to
ii. Flipper flips through a 90 degree angle
iii. Flipper returns to initial position

Alternative Paths:

Postconditions: Flipper is ready to be activated again

Class Diagram Description

View Package

Board interface: The purpose of this interface is to make the methods that the BuildBoard and RunBoard classes will implement as they will be using the same methods.

GBallGui: This works as above although it will be the RunGui and BuildGui that will be implementing this interface.

RunGui: This class is used to create the layout of buttons etc.. for when the user is in the run mode of gizmoball.

BuildGui: This Is used as the same as above although the layout will change as there will be more options for the user to build a unique gizmoball gui.

RunBoard: This is used to paint the gizmos such as lines and bumpers onto the actual gui when in run mode and will be updated using the update method and repaint method.

BuildBoard: This is the same as above although is done in build mode of the game.

GBallView: This is used to create and show the gui depending on what mode the user is in.

SaveGameDialog: This is used to give the user a dialog to specify where to save their game they have built during build mode.

LoadGameDialog: As above although to specify which file to use to load a prebuilt level can this can be done in build or run mode.

NewGizmoDialog: This is for the user to select a type of gizmo they would like to add to the level in build mode i.e. triangle, ball etc…

NewBumperDialog: This is used to get the attributes a user would like associated with a particular bumper i.e. the speed and direction of the bumper.

NewFlipperDialog: As above except with key bind option for user to pick which key is used to activate the flipper.

InformationDialog: This is used to give feedback to the user when doing certain tasks such as adding gizmos.

ErrorMessageDialog: As above except that gives feedback on whether a gizmo can be placed at a certain point.

The above dialogs may change and an interface may even be used if some dialogs are of similar nature.

Model Package

Gizmo interface: This is used as all the gizmos will have similar methods to set the attributes such as position and sizes.

The following classes will implement the Gizmo interface as they will be similar and will have the key attributes that will allow the making and positioning of the gizmos in the game.

- RightFlipper
- LeftFlipper
- Absorber
- SquareBumper
- TriangularBumper
- CircleBumper

Ball: Used similar as above although the ball can move and slow down depending on the velocity and friction also used at the end of lines for squares etc.. with a radius of 0 so that we can get better collisions with gizmos.

Walls: Used to create the positions and thickness for the walls around the play area.

VerticalLines: Used to draw squares, flippers and triangles.

CollisionDetails: this is used to detect collisions between the ball and gizmos and to find the minimum time for the next collision so that the game looks and reacts realistically.

GizmoData: This is used when saving and loading a game and is used to get and set current positions, colours and any other details about gizmos.

Controller Package

The following listeners are connected to the views and listen for changes in the views and then react accordingly using their actionPerformed() method and updates the view when the model changes. It also adds event listeners to the view and updates the model when the user manipulates the view.

AddAbsorberListener: This is called when when in build mode and the user would like add an absorber.

AddBallListener: Used when a user adds a ball to the game.

AddGizmoListener: Used when the user adds a gizmo to the game.

AddLeftFlipperListener: Used when the user adds a left flipper to the game.

AddRightFlipperListener: Used when the user adds a right flipper to the game.

BuildListener: Used when the user is in build mode to determine what actions to take.

RunListener: As above although in run mode.

GizmoBallListener: Used to determine the the speed and velocity off the ball.

GizmoConnectListener: Used to determine if two or more gizmos are connected.

KeyConnectListener: Used to set the key the user has associated with a flipper.

KeyPressListener: Used to determine if an assigned key has been pressed.

LoadFileListener: Used to determine if the user wants to load a game build.

SaveFileListener: Used to determine if the user wants to save a game build.

MouseListener: Used to determine if the mouse has been pressed in add gizmo state for allowing to draw the gizmo.

MoveGizmoListener: Used to determine if the user would like to use a gizmo.

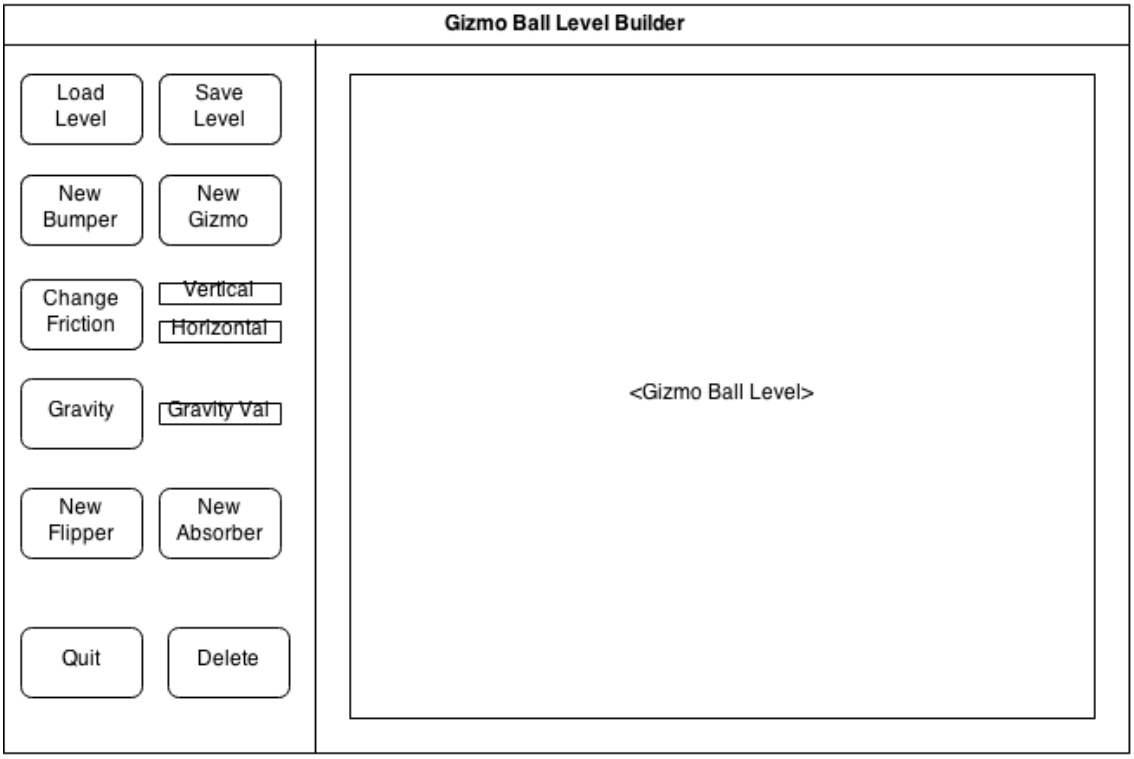
RemoveGizmoListener: Used to determine if the user would like to remove a gizmo.

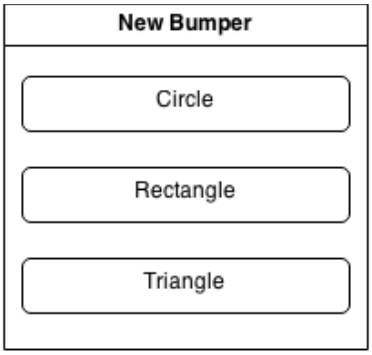
RotateGizmoListener: Used to determine if the user would like to rotate a gizmo.

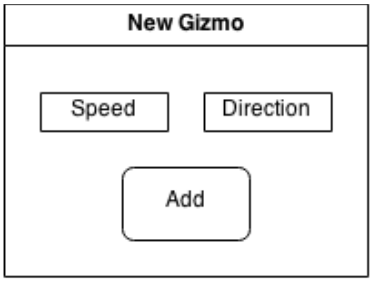
GameExitListener: Used to determine if the user exited the game.

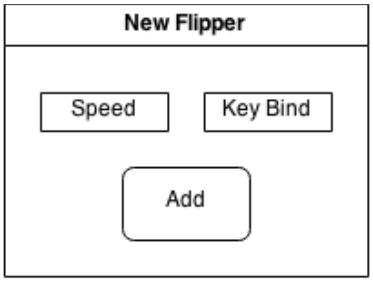
Gui Mockup

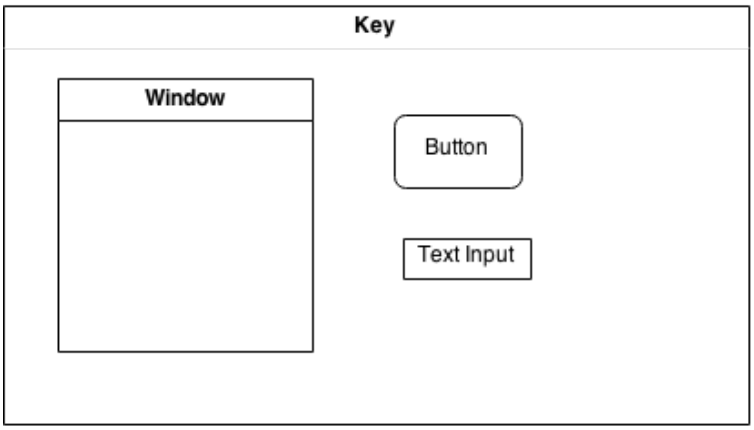

Gizmo Ball Play
Load Level
Level Builder
Start
Stop
High Score
Exit
<Gizmo Ball Level>


Gizmo Ball Level Builder
Load Level
Save Level
New Bumper
New Gizmo
Change Friction
Vertical
Horizontal
Gravity
Gravity Val
New Flipper
New Absorber
Quit
Delete
<Gizmo Ball Level>


New Bumper
Circle
Rectangle
Triangle


New Gizmo
Speed
Direction
Add


New Flipper
Speed
Key Bind
Add


Key
Window
Button
Text Input

Physics Loop

The Physics Loop - how and in what order are ball motion, collision detection, collision resolution, triggering, friction/gravity, and drawing are handled.

The physics loop is the core of the Gizmoball software, it dictates how the ball is going to move around the screen and interact with the Gizmos and the flippers. The ball movement must be as realistic as possible. Therefore, one must take into consideration external forces like friction and gravity into account.
One must keep track of the ball' movement and time until collision of the ball with all other object in the Jpanel (Walls, Balls, Triangles, Circles, Squares, Absorbers, Flippers etc.). This can be done by creating a loop that follows the ball motion every tick and calculates the time until collision using the MIT physics Geometry timeUntilCollision() then if the ball collides with another object one must call a reflect() method in order to calculate the new ball velocity.

for (VerticalLine line : lines) {
time = time to collision
if (time < tick time) {
tick time = time;
new Velocity = reflect form object velocity;

After the collision one must redraw the ball and modify its velocity:

If time until collision > tick time {
no collision

}else {

Move the ball till the collision occur then change the velocity of the ball

}

Then Notify observers ... redraw updated view.

However, there is a catch, the timeUntilCollision()methods assume constant ball velocity . But this is not the case. Therefore, one must take into account the gravity and friction force before or after the "time until / update position / reflect" series of steps. Hence, each tick must apply friction and gravity.

One must of course take into account the flipper movement. Since they move independently of the ball. One must keep track of flippers' motion in order to properly calculate the appropriate time until collision of the ball and the flippers. Furthermore, one must take into consideration the change of velocity when the flippers hit the ball.

One needs to repaint the ball every time the ball collides with an object. However, there might be multiple collisions within each 'tick time'. Therefore, one must not only move each tick time but each *(tick time || minimum time until collision)*.

Triggering System

The triggering system is the system which literally causes the game to run as without the triggering system, the game would not be doing anything to the ball and would be boring and useless. The triggering system allows the ball to change direction velocity and when hitting several different gizmos, it also changes the behaviour and appearance of some of the gizmo's too, for example when a ball hits a gizmo it changes colour. The triggering system can be implemented in two ways: keypress triggers or gizmo-triggers.

Gizmo triggers are implemented in many of the gizmo's, for example in square, circular to triangular bumpers, when a ball hits one of these bumpers the behaviour of the bumper changes which could mean changing colour for example, however these bumpers do not require an action. With flippers, the trigger is generated when a ball hits it meaning that the flippers rotates 90 degrees.

Once the trigger is generated again when the ball hits it again, the flipper returns back to its former position and so on. Absorbers have the same behaviour as the other gizmos, when the ball hits it, its action is to shoot out a stored ball in a straight upward direction back to the top of the playing area. This all depends on whether the absorber is storing a ball because if it is not, or if the previously ejected ball has not yet left the absorber then the absorber takes no action when it receives the trigger signal, so it requires these preconditions. The outer walls can also have a trigger so that when the ball hits it the behaviour changes, such as the ball is projected at a greater velocity for example, however this is not a requirement of the gizmoball specification.

The other main trigger, key-press can be used for less gizmos. The main gizmo that key press could be implemented with is the flippers, so that when a specific key is pressed down the flipper rotates 90 degrees clockwise. When the key is released then the flipper goes back to its former position.

The key press trigger is most desired for the flippers as it gives the user control over the gizmo ball system and allows them to interact with the system, however problems might occur when a key is pressed while the flipper is already rotating, meaning that it is desired for the flipper to respond to all the triggers immediately when it receives the signal, for example if a flipper is in a forward motion and is triggered, it will immediately respond and switch to a backward motion, this would therefore meaning that the system must account for the linear velocity of the flipper that contacts the ball, meaning that the ball may leave the flipper with a higher energy that it had when it had first contact. The stationary bumpers could also implement a key press trigger where the behaviour could be changed but this may be more pointless than having it associated with the flippers.

The triggering system needs to work in accordance with the MITPhysics package so that it gives the game the realistic effect of having gravity working in force, or else it is really kind of pointless and does not give the game a fun aspect which it should do. For example, as

stated earlier, if a ball hits a moving flipper then the ball's velocity may change and become greater than it had been when it initially made contact with the flipper.

Project Plan

Week 3

- All members of the group will finish of the preliminary design for the Wednesday.
- Group discussion on who is doing what in the implementation of the program.
- Create the outline of the project using the class diagrams.

Week 4

- John, Alan and Kyle will start doing the code for the view such as the gui for build mode and run mode this will include the coding of the layouts and screen design using buttons and dialog classes.
- Hristo and Adam will start coding for the controller package doing the listeners.
- As a group we will look over the code done so far and make any changes to the class diagram as the classes will now need to have methods and attributes that the classes are using.
- Various group discussions through the week to see how things are progressing.
- Begin discussions of how to implement the model.

Week 5

- All group members will begin coding of the model package including the coding for gizmos and the triggering system and connecting this to the physics loop as this is probably the most complicated part of the program.
- We will begin to think about Junit testing and validation testing strategy for the model package and how we will go about this. These would be tested based on our use cases.
- Update the project plan and class diagram if needed at this stage.

Week 6

- All group members will work on bringing the project together with the code that has been written so far this will be done using github.
- Finalise the prototypes creating the required jar files.
- All team members will finish of the test, validation and junit testing strategies this will rewritten in a word document so that a user can run through the program and perform the tests specified.

Week 7 and onwards to week 10 finalising the implementation and testing for the finished project this will also include any documentation such as class diagrams and project plan. Implement proposed testing strategies on a users perspective to see the outcome of these.