6.170 – Gizmoball ARCADE Final Design

Team se51

Hyun Soo Kim, Kevin Modzelewski, Albert Ni, Yi Sun

I. REQUIREMENTS

1.1 Overview

Gizmoball ARCADE (also known as GB Arcade) provides users with a 3D environment for designing and playing gizmoball, a game similar to pinball but which also contains a variety of additional features. In addition, GB Arcade comes with a menagerie of arcade-style games, and multiplayer support for certain games.

GB Arcade contains both playing and editing modes. In playing mode, users control various components which they may use to attempt to accomplish various objectives depending on the nature of the specific game in action. In editing mode, users can edit the state of the game world by placing or removing "gizmos", as well as by modifying the properties of these gizmos.

1.2 Revised Specification

Full 3D Movement
GB Arcade supports movement in all three directions and is not constrained to any particular Z-plane. This allows for greater flexibility in possible gameplay, as is seen in the provided MonkeyBall game worlds.

Complete Grid Freedom
Gizmos in GB Arcade do not need to adhere to an integer grid, allowing for the creation of fancier game worlds, as is done in the provided Pinball game worlds.

Absorber Storage
The Absorber gizmo in GB Arcade stores the balls it holds below the Absorber instead of within it. This allows Balls of arbitrary size to be held by the Absorber, since the original specification would require a specific hack to deal with the infinitely occurring collisions that result from storing a Ball larger than the size of the Absorber.

Overlapping Gizmos
While overlapping gizmos are forbidden by default, users are given the option to allow overlapping in order to create fancier game worlds. This is used extensively in the provided game worlds.

1.3 User Manual

1.3.1 Starting GB Arcade

Obtaining GB Arcade requires an active connection to the Internet. Users should open a web browser, direct it to http://web.mit.edu/kmod/www/gizmoball/, and click on the provided link. GB Arcade uses Java Web Start, so users will need to have a Java Runtime Environment (JRE) installed. In addition, only the Windows operating system is officially supported by GB Arcade, though it generally works with other operating systems as well.

1.3.2 The Graphical User Interface

The user starts the game at the welcome screen, which pops up at the center of the screen, with the 'Start New Game', 'Load Saved Game', and 'Quit' buttons. Obviously 'Quit' immediate exists GB Arcade. Before elaborating on the other two buttons, first an introduction of some basic game functions that will appear throughout this section are in order.

1.3.2.1 File Loading

Files can be loaded through the following locations:

Welcome Screen
This can be done simply by pressing 'Load Saved Game'

Play Screen
This can be done by selecting the 'File' menu and choosing 'Load Game' (shortcut Ctrl-L). Activating this brings up a load file dialog. If a game is currently running, it is put into pause mode. Subsequently, there are the following possibilities:

- If the user does not choose a file to load, GB Arcade reverts back to its previous mode.
- If the user chooses a well-formed file, GB Arcade loads the saved game and immediately enters play mode
- If the user chooses a badly formed file, GB Arcade reverts back to its previous mode and displays an error message. No new files are loaded, and aside from the error message, the program proceeds as if nothing happened.

Note that the title bar of the frame changes to reflect a successful load.

1.3.2.2 File Saving

Files can be saved through the following locations:

Play Screen
This can be done simply by selecting the 'File' menu and choosing 'Save Game' (shortcut Ctrl-S), or choosing 'Save Game As…'. Subsequently, there are the following possibilities:

- If there has not been a previous successful save or load, selecting 'Save' brings up a save file dialog. Of course, activating 'Save Game As…' always brings up a save file dialog. If the game is currently in play, the game is put into pause mode.
- If the user does not choose a file to save, GB Arcade reverts back to its previous mode
- If the user chooses a well-formed file, GB Arcade saves the game and returns to play mode
- If the user chooses a badly formed file, GB Arcade reverts back to its previous mode and displays an error message. Aside from the error message, the program proceeds as if nothing happened

Note that the title bar of the frame changes to reflect a successful save. Also, users do not have to include the extension ".xml" in saving, but may choose to do so. For example, saving as "asdf" is the same as saving as "asdf.xml", as both will result in the output being written to "asdf.xml".

1.3.2.3 Starting a New Game

New games can be started through the following locations:

Welcome Screen
This can be done simply by selecting 'Start New Game'

Play Screen

This can be done simply by selecting the ‘File’ menu and choosing ‘Start New Game’ (shortcut Ctrl-N)

1.3.2.4 Entering Editing Mode

Editing mode can be started through the **Play Screen** by selecting the ‘Game Play’ menu and choosing ‘Go to Edit Mode’ (shortcut Alt-E).

Toggling the edit mode brings up a palette screen to the left of the main screen. The palette screen is titled “Palette” and only has the close button available on its frame border. It also has three panels - Gizmos, Properties, and History. The Gizmos panel holds the buttons that allow a user to place Gizmos on the screen. There are a total of seven Gizmos to choose from. From left to right and top to bottom - RectangleBumperGizmo, CircleBumperGizmo, TriangleBumperGizmo, RightFlipper, LeftFlipper, Absorber, and Ball.

Placing Gizmos
The user can place a Gizmo into the world by selecting its corresponding button, which lets the user know that it has been toggled by changing background color to yellow and border color to blue. Then, by clicking on the play screen, users can place a Gizmo at the location of his click. The default z-position of the Gizmo is 0. If Gizmo overlap is disabled, attempting to produce a Gizmo that will overlap with another will produce an error message. There is one exception, as specified by the original specifications, as a ball can be placed within an absorber. If overlapping is chosen to be legal, the user can place any Gizmo anywhere, even outside of the normal game bounds (the walls). Overlapping can be toggled on/off by selecting the ‘Game Play’ menu and navigating to ‘Check for Gizmo overlap’.

The user can un-toggle the create Gizmo option by pressing F5, which refreshes the palette window. Refresh can also be run by going to the Edit menu on the palette window.

Selecting Gizmos
If a user clicks on an existing Gizmo while having a create Gizmo button toggled on, this will still select the Gizmo for editing and ignore the fact that the create Gizmo button is toggled on.

In general, the user can edit a Gizmo by clicking on it in editing mode. The selected object will take on a distinctive texture, notifying the user that it has been selected. Left-clicking on a Gizmo sets the current focus to that Gizmo. Right-clicking does this as well, but also shifts focus to the palette screen.

Editing Gizmos
The palette holds the user's tools. Below is a list of methods at the user's disposal.

Set Position
There are three blank text fields for this purpose, and they are the first triplet of text fields that appear in the Properties panel. The first, second, and third text field from left to right represent the x-coordinate, the y-coordinate, and z-coordinate. The user may edit the text fields and press Apply to apply the changes to the Gizmo.

Note that the Properties screen's border will change to blue if any change is made to the position text fields or they just received focus. The blue border indicates that the fields may have been changed, and that what appears on the field may not reflect the properties of the selected Gizmo.

In order to see the properties of the selected Gizmo, the user should select the 'Edit' menu and choose Refresh.

For an easy way to change Gizmo positions, set the focus to one of the text fields and press up or down. The values will increment or decrement by 0.5 and apply immediately.

Set Velocity
Setting the velocity is enabled only for movable objects. In GB Arcade, the only movable objects are Balls, so only Balls can have their velocities edited. There are three blank text fields for this purpose, and they are the second triplet of text fields that appear in the Properties panel. The first, second, and third text field from left to right represent the x-velocity, the y- velocity, and z- velocity.

Set Coefficient
The user is also permitted to set the coefficient of reflection of a GameObject, and can increment or decrement the value by 0.1 by pressing up or down with the current focus on the corresponding text field.

Rotate
Clicking this button rotates the selected Gizmo clockwise by 90 degrees.

Set Texture
Selecting a texture from the drop-down list alone does not do anything. The user must also press Apply to apply the texture to the Gizmo. If the currently selected texture is "custom

color", then the texture of the Gizmo is a plain color, determined by the last color that was set or accessed.

Set Key Assignments

Clicking this button brings up two splash screens, each asking for user input. The first screen records the user's key input to link the key pressing action to triggering the current Gizmo. The second screen records the user's key input to link the key releasing action to triggering the current Gizmo.

Note that repeating a key that already has an associated trigger deletes that key's assignment to the specified Gizmo. Also note that pressing any key will register as an input. There are two ways to get out of the Set Key process without making any changes. The first way is to lose focus of the window, such as by clicking something else. The other way is to click the window, which is programmed to close.

Features in a nutshell

Editing

- Left-clicking on a Gizmo allows user to focus to edit that Gizmo
- Right-clicking on a Gizmo in addition returns focus to the palette
- Once a user edits a certain editable component, that component is stored. Whenever the user right-clicks on another Gizmo (recall that right-clicking on a Gizmo returns focus to the palette), that saved component receives focus. Hence, if the user is in the "editing coordinates" mood, he can simply right-click on a Gizmo, use arrows to move it (since the focus is on the text field), right-click another Gizmo, use arrows to move it, etc.
- Once a user gets focus on an editable component or edits a field, the Properties panel's border turns blue. This feature is to let the user know that the values in the fields do not necessarily reflect that of the object selected. In order to get the object's traits, press Refresh (F5). If the border is not blue, the user is guaranteed that the selected Gizmo has the properties listed in the properties panel
- Refresh (F5) also untoggles the Gizmo that is to be placed on the screen. That is, if one is about to create a Gizmo and clicks on "Left", for example, he can toggle out of creation mode by pressing F5
- If no gizmo is focused on, the input fields will be disabled, as well as the Deselect button
- If a gizmo is focused, appropriate input fields will be enabled
- The user is able to edit the texture that is on the Gizmo with a drop-down combo box
- The user is able to edit the coefficient of reflection of the Gizmo
- The user is able to edit the position of the Gizmo
- The user is able to edit the velocity of the Gizmo, if the Gizmo is a ball

- The user is able to rotate the Gizmo 90 degrees clockwise (from user's view)
- The user is able to edit the key that activates the Gizmo (if that Gizmo is not a ball)
- The user is able to remove the key that activates the Gizmo (simply type the same key to remove the mapping)
- The user can specify whether he wants key down or key up to affect the trigger
- When the user adds a Gizmo to the world, it is selected as the current Gizmo to be edited
- When the user removes a Gizmo, the Gizmo focus is lost
- The KeyInputScreen first displays a window asking the user to type in a key to map key pressed events. Any key the user presses will be read by the program. The user can get out of the key-setting process entirely by taking focus of another window. If the user clicks on the key window, a second window will pop up requesting the user to map key released events
- After typing a key, a message dialog shows up confirming the key
- If any of the text fields have invalid values (i.e., alphabet characters where numbers are required) an error message comes up
- Closing the palette screen returns the game to play mode
- The palette screen, when it's invoked when editing mode is toggled, shows up exactly to the left of the play screen

History
- The user should take advantage of the History list, which keeps track of changes made to the game world so far. It is cleared upon returning to play mode or upon clicking Clear button below it. Users can undo an action by either double clicking on the last valid action (valid actions are in bold, undone actions are in faint italics) or going to the Edit menu and clicking undo. If actions 1, 2, 3, 4, 5 are done in the given order, then double-clicking on 3 undoes actions 3, 4, and 5. Double-clicking on 4 then does actions 3 and 4. Performing an action at this point will get rid of 5, the undone action, and replace it with the new action. In general, performing an action will remove all undone actions and do the new one
- The action history display is a scroll pane with actions as elements. It scrolls automatically with respect to the last action that was done. If the actions are undone, the scroll will move so that the most recently done action shows up at the top
- Purge removes all undone actions from the history list
- Clear removes all actions from the history list
- For the user's convenience, the position of the vertical scrollbar depends on the most recently done action
- When an action is undone, then the Gizmo to which the action was undone is focused, unless that Gizmo has been deleted, in which case focus is lost
- When an action is redone, then the Gizmo to which the action was redone is focused, unless that Gizmo has been deleted, in which case focus is lost

General

- The menu item ‘Hot keys…’ displays a list of keys that have triggers associated with them as well as the Gizmo that they are connected to
- Camera settings can be altered. The user can move in one direction at most three times. Reset camera resets the camera view to the original position
- Camera can be moved in any mode
- Hide Wall Wireframe hides the wireframe of the game’s bounding walls while Show Wall Wireframe shows them
- Pausing the game does not affect any state of the game except for the fact that it stops running
- No key inputs will trigger anything if the game is paused
- Selecting File and then Return to Main Menu will prompt the user whether he really wants to do so
- Selecting File and then Quit will prompt the user whether he really wants to do so

1.3.3 Playing Mode

In addition to custom game worlds created by users, GB Arcade also comes with a variety of preset games that can be played to a user’s heart’s desire!

Multiplayer Support

Many of the provided game worlds include multiplayer functionality so that users can play with their friends. In addition, the MonkeyBall and SpaceInvaders game worlds support networked multiplayer so users can even play with people on other computers. Upon startup of games with networked multiplayer support, GB Arcade asks the user if they would like to play multiplayer. If the user chooses multiplayer, they can elect to be the server or the client for the game. If server is selected, the user should give a friend the server’s hostname/IP address to connect to. Similarly, if client is selected, the user must provide the hostname/IP address of the server to connect to.

Demo

This game world simply displays some of the functionality of the GB Arcade system. For instance, delays exist in various Gizmos, including the Flipper, which has a delay of 0.25 seconds, and the Absorber on the bottom, which has a delay of 1 second. Additionally, Demo includes a small Teleporter below the Flipper which teleports Balls back to their original position. Finally, the camera can be set to rotate around the scene.

Each of the five Balls have associated Triggers which set the Color of anything a Ball hits to be the same Color as the Ball itself. Thus, when two Balls collide, they switch colors.

Hotkeys (note that hotkeys are CASE sensitive)
a Flips the flipper
r Toggles camera rotation

Breakout

The object of Breakout is to break all of the blocks without allowing the Ball to hit the bottom of the screen. Players prevent this from happening by moving the provided Paddle left and right to reflect the Ball. In addition, this Paddle can be used to aim the Ball since it is curved. In the traditional Breakout game, players lose a life when the Ball hits the bottom, though in this game it simply bounces off of the bottom. There is also a box at the very top of the map which contains another Ball bouncing around inside it. If this box is struck, the other Ball is released, thus doubling the player's block-breaking potential.

Hotkeys
z Moves the paddle to the left.
/ Moves the paddle to the right.

MonkeyBall

MonkeyBall is similar in spirit to the classic hand-held Labyrinth game where players tilt the board to move the ball around while trying to avoid the holes. However, with MonkeyBall the Ball is being moved around in full 3D environments by tilting the world to navigate the Ball to the goal. In addition, players can also cause the Ball to jump, which helps them to reach farther locations and in some cases makes some shortcuts possible. The provided worlds include multiple 3D elements, including rotating gizmos. The camera is set to automatically follow the Ball, and there are multiple levels.

Hotkeys
Arrows These tilt the world, so that the ball will accelerate in the direction pressed

Multiplayer support
After connecting to another player, the server chooses which level to play. Both players then play the level independently, but can also see a "Ghost Ball" which indicates where the other player's Ball currently is. While there is no limit to how many people can watch each other player, currently the limit is set to be 2.

Pong

This is the classic game of pong, but with a fresh twist! Now instead of having flat Paddles, they are slightly rounded, meaning shots can be directed. Also, since the physics are realistic, if a

player's Paddle is moving towards the Ball, the Ball is reflected with a higher velocity than if the Paddle is moving away from the Ball.

Hotkeys

a	Player 1 up
z	Player 1 down
up	Player 2 up
down	Player 2 down

Multiplayer support
Two people play on the same computer against each other

Slime Volleyball

In this game, players control a small blob known as a Slime which slides around on the field. Each player aims to get the (volley)ball to land on their opponent's side while preventing it from landing on their own side. Players can move around and jump, and control the Ball by striking it with their Slime.

Hotkeys

w	Player 1 up
a	Player 1 left
d	Player 1 right
i	Player 2 up
j	Player 2 left
l	Player 2 right

Multiplayer support
Both people play on the same computer against each other, trying to get the ball to land on the other side.

Space Invaders
The goal of this classic is to survive and kill as many enemy ships as possible. Players receive points for killing enemy ships, and lose points for dying. These points can be used to buy power ups, and in multiplayer network mode, cause bad things to happen to an opponent. Each player has a corresponding score, which is the sum of the points gained through killing enemies minus the points lost through dying or from buying items. The following power ups are available for purchase:

Shields – Shields protect a player's ship from attack. Players start with one shield by default.
Fire Rate – Fire Rate power ups enable a player's ship to shoot faster
Guns – Guns increase the number of shots fired at a time. A player can have up to 5 guns.
Enemy Ships – In multiplayer mode, players may purchase enemy ships which will then appear on their opponent's screen to hopefully overwhelm them. These purchased enemy ships are

distinguished by being yellow, but are otherwise behaviorally equivalent to the normal enemy ships.

Hotkeys

left	Moves the ship left
right	Moves the ship right
space	Fires the ship's gun (this can be held down for continuous fire, limited by the ship's fire rate)
f	Purchases a firing rate increase of 10% (cost 500 points)
t	Sends one enemy ship to the opponent (cost 75 points)
r	Sends ten enemy ships to the opponent (cost 600 points)
s	Purchase a single-use shield (cost 300 points), note that shields stack
g	Purchases a new bullet gun (cost 1000 points)
b	Buys a new beam gun (cost 1000 points)

Note that the cost of dying is 500 points.

Multiplayer support

After connecting to a friend, both games will start to play simultaneously. Each player can see their own score as well as their opponent's score at all times

1.3.4 Saving and Loading

1.3.4.1 Saving

GB Arcade provides users with the ability to save games so that they can be loaded and continued later. Maps can be saved by going to the 'File' menu and selecting 'Save Game' (hotkey Ctrl-S) or 'Save Game As…'.

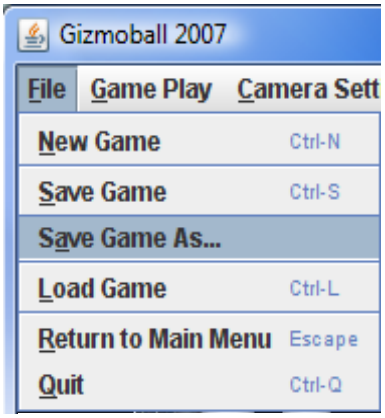


Figure: Save Game As in the File menu

If the current game has yet to be previously saved, or if 'Save Game As' is selected, users are able to specify the name and location of this file. Games are saved in the ".xml" file format.

1.3.4.2 Loading

Users may load XML files corresponding to previously saved GB Arcade games. Such files are first validated against a schema, which checks that the file is in a format that GB Arcade can properly process, and an error message pops up in the event that the file does not actually validate.

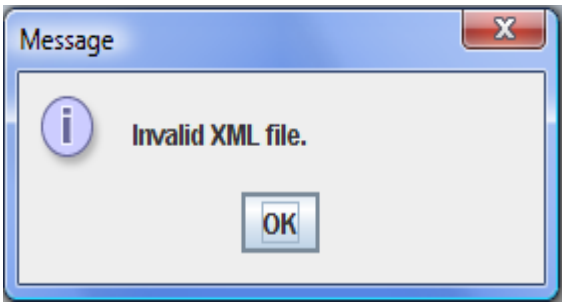


Figure: When users attempt to load an invalid XML file, GB Arcade produces this message

Any game world created solely through the use of Editing Mode can be saved and then loaded exactly

in its previous form. However, users should be cautioned that while it is possible to save games that were initialized through the use of one of the provided game worlds, in almost all such cases some of the game information will fail to be properly saved. Consequently, the game world produced upon reloading will not be completely the same as the game world used to save it.

Additionally, since games are saved in the highly versatile XML file format, these files can be transferred from computer to computer and even run through other gizmoball systems. Of course, due to variations in the implementation of these systems, it is highly unlikely that loading an XML file through another gizmoball game will produce anything near the same thing as loading it through GB Arcade.

1.4 Performance

In general, GB Arcade runs smoothly at 50 frames per second on a 2.0 GHz machine with 256 MB of main memory and properly installed video drivers. However, it is conceivable for users to create game worlds with enough gizmo balls so that this frame rate is no longer obtainable.

Additionally, there currently exists an unresolved performance issue where multiplayer mode has caused a significant slowdown of the entire system. Attempts thus far at deterministically reproducing this problem have failed.

1.5 Problem Analysis

This section outlines the major conceptual obstacles that any gizmoball system, and specifically GB Arcade, needs to be able to overcome.

1.5.1 Physics

1.5.1.1 Required Functionality

Fundamentally, the physics package used in GB Arcade needs to support three basic capabilities:

1) Physics Model Creation

Primitive types of physics objects such as spheres, plane circles, plane polygons, lateral cylinders, and tori must be supported in order to draw basic shapes. In addition, they must be able to combine these shapes into larger rigid bodies (physics models) to compose the structure of the gizmos.

2) Physics Model Movement

Objects must be able to move according to the laws of physics, namely gravity and friction, in the absence of external forces.

3) Collision Detection and Response

The system must be able to determine when a collision between a ball and any other object in the system will happen, and respond to the situation appropriately.

1.5.1.2 Achievement of Required Functionality

The system accomplishes these goals in the following way:

Goal 1 - Each primitive physics object is represented by a PhysicsObject abstract data type. PhysicsObjects abstracts the basic functionality of a simple physical object. PhysicsObjects can be put together to form more complicated PhysicsModels, which are the physical representations of things such as gizmos. Note here that balls are represented physically by PhysicsSphere, a type of PhysicsObject, since it is necessary to know more specific information about it than the generic fields of a physics model will allow. By implementing all physics capabilities in terms of the basic operations common to all PhysicsObjects by extending those to PhysicsModels, the physics system allows for creation and use of very complicated configurations without any need to think about or deal with them separately. The calculation of bounding spheres of PhysicsModels, for example, is done automatically from the smaller bounding spheres of each PhysicsObject in them.

In addition, PhysicsModels encapsulate the addition of basic geometric shapes in a manner that includes the additional edges and points that are needed for collision detection. For instance, adding a rectangular box with the edges and points is supported, allowing for greater ease of use and more realistic collision detection.

Goal 2 - PhysicsModels support rotation and translation by delegating to the appropriate behavior in each PhysicsObject. The physics package supports movement in the absence of external forces, and by allowing the velocity of objects to be set, it allows for external handling of gravity and friction.

Goal 3 - Collision detection and handling are performed separately in the system. At each time step, the physics engine detects the next collision by finding the minimum time for each ball to collide with the PhysicsModel of each other GameObject. Each PhysicsModel handles detecting collisions between any ball and itself by delegating to the appropriate methods within its constituent PhysicsObjects.

After collisions are detected, the game is updated until the collision, and it is then processed to change the velocity of the ball based on the collision processing of each PhysicsModel, which

again delegate to the constituent PhysicsObjects. Note that in this process, it is imperative that every collision in the system involves at least on ball, since this means that collision detection and handling only need to be done between the primitive PhysicsObjects and balls.

Bounding spheres are used in this collision detection algorithm to optimize collision detection. Each PhysicsModel has a bounding sphere that moves with it, and if the ball will not be inside the bounding sphere before the collision time, the ball is not checked against the PhysicsObjects in that PhysicsModel.

1.5.2 World Simulation

Graphically, the system must actually be able to keep up with and display the game world. This requires going forward in time and calculating how the game progresses, from both a physics perspective (described above) and from a graphical perspective. Since it is possible for a ball to move 400 times its diameter in one second, it is necessary to be able to handle a wide spectrum of possible interactions.

From the highest level perspective, the game world is processed time step to time step. Within each time step, the system first determines the time until the first collision. If this time is larger than the time step, the system simply updates all of the objects in the game by the time step. Otherwise, all of the objects in the game are updated by the time until the first collision, the effects of the collision are taken into account, and then the processing of this time step continues. In addition, user input, such as key presses, that occurs within a specific time step, is stored and processed before the next time step. Since each time step is 1/50 of a second, this slight delay in response is completely unnoticeable.

The main issue in world simulation (other than the physics routines) is how to process the effects of a collision. This requires functionality for storing actions, as well as which actions should be executed as a result of specific collisions, and is done through the use of a Trigger system.

1.5.3 Graphical User Interface

The main challenge in creating an effective GUI was to integrate a number of windows to work with each other with regard to an underlying data structure. A number of windows may be open at the same time during the Editing mode, but there will not be multiple copies of the state of the game. The windows would be responding to the same set of data.

Another problem was in threading the application correctly. There should be only one thread running the game in question, but there are actually multiple threads working at once, since

user inputs correspond to extra threads. This leads to some synchronization issues if a user input and game runner happen to edit the same piece of data.

There was also the problem of effectively taking in user inputs and temporarily storing and then altering them.

1.5.4 The XML File Format

1.5.4.1 Saving

Games must be saved into an XML file format that will validate against a provided schema. In addition, optimally the XML files should also be able to store as much information pertaining to the unique aspects of GB Arcade as possible.

To make all XML files validate against the provided schema, for the specific types of Gizmos corresponding to the special cases within the schema, the required attributes are approximated as best as possible. Such an approximation is necessary in some cases since the provided schema has certain restrictions such as integer coordinates for the gizmos, and multiples of 90 degrees for the rotation, while GB Arcade supports more general positioning.

As for storing information specific to GB Arcade, the XML Parser uses a general strategy of storing the inputs to the constructor of a Gizmo and tagging them appropriately. In addition, other attributes of Gizmos unique to GB Arcade, such as color, texture, etc., are also stored with specifically pre-defined tags.

1.5.4.2 Loading

When parsing each line of an XML file, the parser checks for tags indicating that specific line was generated by the GB Arcade XML writer. If such tags are found, the parser simply parses the tags corresponding to the constructor of the Gizmo for that line and feeds them into the constructor. However, if no such tags are found, thus indicating the line was generated by some other implementation of gizmo ball's XML writer, the parser then resorts to using the default attributes corresponding to the provided schema to generate the Gizmo.

II. Design

2.1 Overview

At the highest level, GB Arcade consists of the following major components – Physics, Graphics, Game World, User Interface, Editor, and XML Parser. These components do not strictly

correspond to specific classes, but rather sets of classes. While these parts of the system are not all completely disjoint, when interfaces between two of the aforementioned components do exist, they are simple enough that the implementer of one needs to know very little about the design of the other. The following subsections of this Overview section of the Design describes the major components with respect to the level of interaction between them.

2.1.1 Physics

The Physics portion is the most independent, as for all intents and purposes it is capable of standing alone. Naturally, Physics handles the physics aspects of interactions within the game. Helping to facilitate this process is concept of the five physics primitives – spheres, lateral cylinders, plane polygons, plane circles, and tori, of which every single object that can be found in the game is composed. Thus, processing physical interactions between different objects is reduced to processing physical interactions between physics primitives, which substantially simplifies things. In addition, these primitives serve as the key to the interface between Physics and the "outside world", as the rest of the system need only be aware of the existence and properties of physics primitives.

2.1.2 Graphics

The Graphics module draws all the objects in the 3D world. Although it uses the given physics primitives, aside from that it is completely separate from physics and the rest of the game world.

2.1.3 Game World

Encapsulating Physics and Graphics for the purposes of GB Arcade is a Game World. Game Worlds consist of both the physical objects within a game, as well as the other aspects required for full functionality such as triggers. Objects must obey the laws of physics corresponding to the parameters of a specific game world, and be displayed properly. Obeying the laws of physics simply requires knowledge of the physics primitives, since an object can then select the proper combination of primitives to compose it and let these primitives handle physical interactions. Displaying objects then follows, since the Graphics portion of the system also uses physics primitives to produce the proper visuals.

2.1.4 User Interface

The User Interface converts the Game World from a collection of abstract concepts to a tangible cohesive physical display. Additionally, it provides an interface through which users can interact with a Game World.

2.1.5 Editor

The Editor uses the User Interface and knowledge of the various components that can go into a Game World to provide an interface through which users can manipulate a Game World.

2.1.6 XML Parser

The XML Parser is unique in that for the most part, it has very little effect on the user experience. However, it provides the functionality for saving and loading games, meaning it collects all of the information needed to convert a Game World into a format that it also knows how to process and restore into a Game World. Thus, the XML Parser is in essence the end product of the process of converting abstract concepts into a tangible and quantifiable form.

2.1.7 Continuation

This rest of this section aims to provide a fairly in-depth view of the design of the GB Arcade system. To do so, two different perspectives will be used to examine the system's structure. The first perspective focuses on the low-level "building blocks" of the system. The second perspective takes a more top-down view to show how the system puts everything together.

2.2 Low Level View

GB Arcade can be thought of as having two fundamental building blocks, namely GameObjects and Triggers. Technically, GameObjects are actually composed of other, smaller pieces, namely physics primitives. However, these smaller pieces are effectively completely encapsulated within GameObjects as they have no impact on the system except through the GameObjects that they compose. Thus, from a design perspective it is sufficient to understand the composition of GameObjects, and then consider everything else in terms of GameObjects and Triggers.

2.2.1 GameObject

At the highest level, every physical thing that can be found in a game world is a GameObject, and more specifically a Ball or a Gizmo. From an implementation perspective, all GameObjects contain two major things - a GraphicsProperties object which encapsulates the information needed for drawing an object, and a list of default triggers which are always associated with specific instances GameObjects. For instance, since collisions are handled through the use of a SimpleCollisionTrigger, SimpleCollisionTrigger is a default trigger for balls.

GraphicsProperties can be directly created directly from its corresponding GameObject's physical model (and this is how it is almost always done in the system), but it leaves open the possibility of an object being drawn differently than its actual physical representation in the

game world. One application of this is that it allows the creation of objects that appear to pass through each other without modifying the collisions framework of the system.

The appearance of an object can be specified either through a texture file or a color. Since colors can be created with a level of transparency, objects can as well. Along these lines, objects can (and in many instances of the provided GB Arcade game worlds) be created to be invisible, or to be drawn in wireframe.

Graphics is split into three main classes. First, TextureHandler takes care of creating, loading, and applying textures. Second, PrimitivesDrawer takes care of actually drawing the physics primitives on the screen. Third, WorldDrawer pulls everything together, handles camera movement and has the main drawing function.

2.2.1.1 Ball

While Balls are in many ways very similar to Gizmos, an explicit distinction is made for three primary reasons, and a variety of other minor reasons. First, Balls are not only mobile, but have absolutely no anchor, and can in theory travel to any point in a game world. This is a property that no Gizmo has. Because of this, Balls contain internal state for velocity, gravity, and friction, all of which Gizmos never have. Second, every collision involves at least one Ball, meaning that collision checking is always done by iterating through the Balls in the game and checking for collisions between those Balls and other objects. Third, Balls are restricted to being spheres, so they are composed solely of a PhysicsSphere physics primitive and thus unlike Gizmos, do not contain an entire PhysicsModel.

2.2.1.2 Gizmo

All specific Gizmos extend the Gizmo.java abstract class. Internally, Gizmos contain a PhysicsModel, which is a collection of the five physics primitives, to represent its physical properties. Each specific Gizmo contains the appropriate constructors and mutators to provide the desired functionality. Several mutators are common to all Gizmos and can be found in the abstract class, namely mutators for position, color, texture, coefficient of reflection, rotation, and delay.

The following is a set of the most common Gizmos that can be found in the GB Arcade world. Note that some of the more complex Gizmos are not currently supported in the editor, but will be added at a later date.

BUMPERS

Bumpers are standard objects in the playing field that Balls will reflect off of, and have no other functionality from a strict physics perspective. However, Bumpers will commonly have associated Triggers which are set into action whenever a Ball collides into that a Bumper.

Shapes that Bumpers take on include spheres, cylinders, cubes, rectangular and triangular prisms, and tori.

WALLS

The Walls surround the game, are immovable and never change. While Walls are technically gizmos, by convention they have no associated triggers or anything else, and are invisible by default. However, Walls can be toggled to be "visible", in which case a wireframe of the walls is displayed.

FLIPPERS

Flippers can be triggered through an assigned keystroke to rotate 90 degrees. The speed of this rotation is 6π radians per second. When the key is released, flippers rotate back to their natural state.

BALL SHOOTERS

BallShooters are Gizmos that absorb balls instead of reflecting balls that collide with them, and shoot them out later when triggered. There are two types of BallShooters in GB Arcade – Absorbers, and Teleporters.

ABSORBERS

Absorbers are Gizmos that absorb instead of reflect balls, store them, and shoot them out from one end of the Absorber. They are typically placed at the bottom of the game world to catch balls that have fallen down due to gravity.

TELEPORTERS

Teleporters are gizmos that take in balls and shoot them out from some other location, and can be found in several of the provided GB Arcade game worlds. Teleporters are currently unsupported in the editor.

DETECTOR PLANE

DetectorPlanes are polygonal planes that detect the passage of a ball through them in a certain general direction, and can be found in several of the provided GB Arcade game worlds. DetectorPlanes are currently unsupported in the editor.

LIMITED GIZMOS

Some Gizmos in the system have limited versions (LimitedAbsorber and LimitedTeleporter) which are associated with some counter. These will only function as long as the counter is positive and are disabled afterwards. LimitedGizmos are currently unsupported in the editor.

2.2.2 Trigger

Triggers are used to encapsulate responses in the game world to collisions. Each trigger is defined by two GameObjects – the GameObject for which a collision between a Ball and that GameObject cause the Trigger to be activated in the first place, and the GameObject that is affected as a result. Note that while technically the two objects are considered as the "causer" of the Trigger and the "receiver" of its effects, it is certainly possible for these two GameObjects to be the same, as is it possible for both GameObjects to be affected. All of this is done through the *run()* method in each Trigger, which contains the game world's response logic specific to that Trigger.

For instance, a SimpleCollisionTrigger is used to handle the modification of a ball's velocity upon that ball's collision with an object. However, Triggers can become quite complex, with other possible functionality including GameObject removal and adding points to the score of a game. In fact, the Trigger system is so flexible that it is even used to process key pressed. More specifically, when a key is pressed or released, it causes a set of Triggers containing logic for handling that key press to run.

Due to the common attributes of Gizmos and/or GameObjects, a variety of Triggers exist that work for any GameObject. For instance, since all Gizmos have an *activate()* method, the ActivateGizmoTrigger can work for any Gizmo. However, there are other cases in which a Trigger was specifically designed for a certain Gizmo. An example of this is the getMoveLeftTrigger which returns a trigger that can move a Paddle, as described in the Breakout section earlier, to the left. Since only Paddles have the functionality for moving left, getMoveLeftTrigger does not function with other types of Gizmos.

The following is a set of some of the most commonly found Triggers in the GB Arcade system. It should be noted that in some cases, it is possible for two Triggers to have conflicting actions, such as with AbsorbBallTrigger and SimpleCollisionTrigger. To handle this, different Triggers are assigned a priority, which is used to determine which Trigger takes precedence in the event of a conflict.

AbsorbBallTrigger

An AbsorbBallTrigger handles the functionality for absorbing a ball into a given BallShooter instead of having the Ball collide into it. Note that this implies that AbsorbBallTrigger has a higher priority than SimpleCollisionTrigger.

ActivateGizmoTrigger

An ActivateGizmoTrigger activates a given Gizmo.

CounterTrigger

A CounterTrigger changes the value of a Counter, which is an object used to store a specific count for a game, by the given increment.

NoCollisionTrigger

A NoCollisionTrigger reverses the effects of a collision

SetColorTrigger

A SetColorTrigger sets the color of the collided object to that of the colliding object.

SimpleCollisionTrigger

A SimpleCollisionTrigger provides the default functionality for collision handling.

ToggleColorTrigger

A ToggleColorTrigger switches the color of a Gizmo depending on its current color.

ToggleTrigger

A ToggleTrigger toggles an underlying ToggleGizmo when it activates.

GroupTrigger

A GroupTrigger wraps an underlying trigger to activate only if each of a set of ToggleGizmos is on. It also runs a set of triggers to reset the state of the ToggleGizmos after it is activated.

2.3 High Level View

The GB Arcade system is run through the GamePlay class. Code-wise, GamePlay.java is quite simple, as it basically just branches into the different major components, GamePlayModel, PaletteModel, and ScreenManager, that make GB Arcade work. However, the GB Arcade coding structure is highly modular, with each of these major components being almost completely separated from one another. Thus, it is fair to consider GB Arcade the “central figure” of the software system.

2.3.1 GamePlayModel

Everything that is needed for Playing Mode of the game to function properly works through GamePlayModel. Most importantly, GamePlayModel contains and handles a **WorldModel** and **GameRunner**, each of which is described in further detail below. Additionally, GamePlayModel handles some a smattering of other components that involve the running of the game itself, gizmo interactions, etc. One example of such an aspect of the game is the frame rate, which is only a concern when the game is actually running. Another example is the gizmo overlap Boolean, which toggles whether or not gizmo overlaps are allowed, since this Boolean only has an effect on the gizmos of the game.

2.3.1.2 WorldModel

WorldModel acts as a grand container for all the components that make a game run, though it knows nothing about how to actually use these to do anything. The most important component of a WorldModel is the **WorldState**. In addition, WorldModel contains information about hotkey triggers, players, and other components that aren't explicitly some form of GameObject or Trigger between Gizmos.

WorldState

WorldState represents the end of the first top-down branching path. WorldState contains all of the GameObjects, as well as all Triggers between pairs of Gizmos.

2.3.1.3 GameRunner

GameRunner acts as the "heartbeat" of a given WorldState, and performs the calculations necessary for the advancement of the game in Playing mode one time step at a time. Such calculations include collision calculations, trigger calculations, and object position update calculations.

2.3.2 User Interface

2.3.2.1 Design Overview

A variant of the Model View Controller was used to implement the GUI. While the aforementioned GamePlayModel does not know about the existence of PaletteModel, PaletteModel observes GamePlayModel and maintains a pointer to it. PaletteModel also holds all of the information necessary to edit the game.

Both GamePlayModel and PaletteModel extend Model, an abstract class containing practically nothing except for the fact that it implements Observable. Preliminary design indicated that

there might be considerable overlap between the two Models, but this turned out not to be the case.

GamePlayModel contains a reference to the WorldModel, the key-to-trigger mapping, and camera position information. It also holds information about the last file that was saved or loaded. It also knows of the current mode of the game, whether it is playing, paused, in edit mode, or welcome mode. It is also aware of what keys have currently been pressed down.

WorldModel is basically a wrapper class for WorldState. WorldState can be understood as the minimum data structure required to load a game, but ignores the existence of user inputs. WorldModel stores a key mapping that can be altered to acknowledge user input.

2.3.2.2 Runtime Structure

PaletteModel uses its reference to GamePlayModel and refers to it for information necessary about the game. It mainly uses the GamePlayModel to get access to the WorldModel and to pass it to methods and classes that require it. It also holds variables that allow state functions to exist, such as the allowOverlap boolean value that stores whether the user allows overlapping during the editing mode. It also keeps track of all of the Gizmo edits and key mapping edits done by the user, so that he can undo and redo. It creates and utilizes EditActions and stores a list of them.

ScreenManager initializes and keeps references to all of the Screens in use in the game. There is exactly one copy of each Screen. ScreenManager initializes the observers (Screens) so that PlayScreen observes GamePlayModel and PaletteModel. WelcomeScreen observes GamePlayModel. PaletteScreen observes GamePlayModel and PaletteModel. KeyInputScreen is used to receive user key input about what keys he wants to link to Gizmos. We deviate slightly from the MVC model implemented by Java here. Java has the update method for Observers take in an Observable ob. However, for something observing two things, we would like to take in both object's information when updating the observer. Hence, we have copies of GamePlayModel and PaletteModel in ScreenManager that the screens have access to. The Screens do not know about the existence of each other, and the Models do not know about the existence of the Screens.

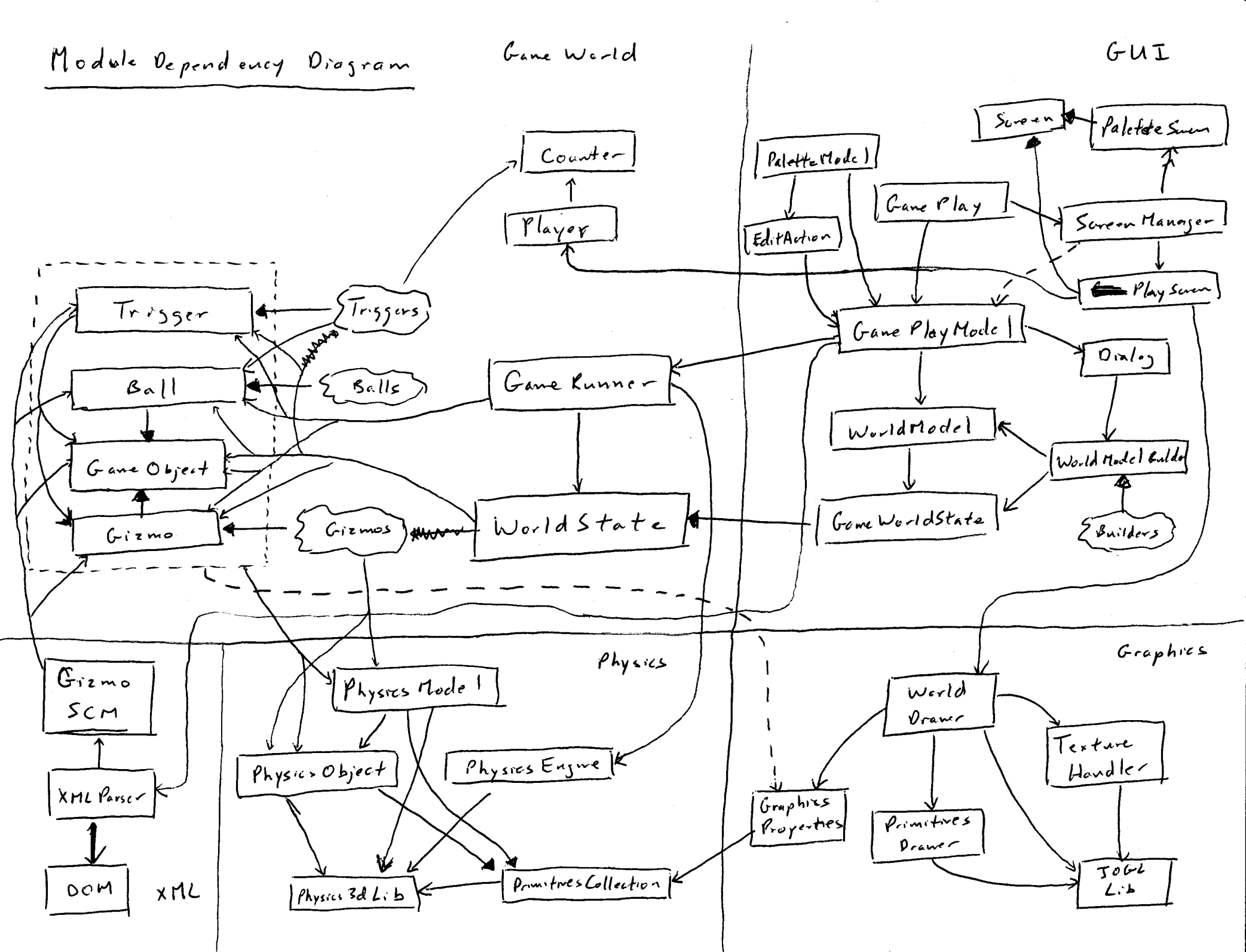
Dialog is a factory class that generates Dialog windows to notify the user of events and also get feedback.

EditAction is an abstract class that serves as the base for instantiable EditActions such as KeyMappingEditAction. It necessitates the existence of the method invert() in all of its children, so that any user action can be undone. GameObjectAddRemoveAction is used to remove or add Gizmos to the world. GameObjectEditAction is used to alter properties of a Gizmo, but not remove or add a Gizmo. KeyMappingEditAction is used to add or remove keys linked to Gizmos.

KeyableAction holds information about actions that users can trigger during the game.
Properties holds the editable features of a Gizmo. It contains a function apply which makes the argument Gizmo take on the properties within. It contains the clone function that allows it to make copies of itself for editing and use.

2.3.3 Module Dependency Diagram

Module Dependency Diagram



III. Testing

3.1 Strategy

3.1.1 Physics

Testing for the physics module took place through three different forms – Unit Testing, checkReps, and Stress Tests

3.1.1.1 Unit Tests

While the difficulty of determining the correct answers for most collision and movement methods precludes the usage of too many unit tests, unit tests were still used to determine the correctness of the rotational collision methods.

3.1.1.2 checkReps

Extremely thorough representation invariants for each physics class were written, and checkRep() assertions were used at the beginning and end of each public method in order to ensure that no representation invariants are violated.

3.1.1.3 Stress Tests

The physics engine was observed for anomalous behavior during repeated gameplay. Anomalous behavior was in many parts defined by what was usually observed, due to the difficulty of otherwise defining what sort of behavior should be expected.

3.1.2 Graphics

The graphics system is almost impossible to automatically test. Hence, it was implemented one feature at a time, and analyzed after the addition of each new feature. At first, the system would only draw polygons. Once this was determined to be working as expected, spheres were added, then plane circles, and so on. Soon, Graphics and the rest of the system reached a state in which testing could be done through the creation of specific test levels involving the latest added feature.

3.1.3 Game World

The components of the Game World are roughly in the middle of the system, as it depends on the graphics and physics systems to run properly, and is run by the GUI. Once graphics and physics reached a basic level, a stub was written for the state of the world, as well as a testbed for testing. This allowed for the testing of features of the Game World, such as Gizmos and Triggers, without requiring a working GUI. Since it is nearly impossible to specify exactly what

the "correct" behavior of the game world is, testing consisted primarily of setting up situations in which the expected outcome is known, and then observing the resulting behavior.

3.1.4 Graphical User Interface

The main strategy used was to make sure that the functions did what they were expected to do, which was not hard.

For the menu bar, we simply clicked on each item under various circumstances and verified their functionalities. It was not too tedious since there was only a small number of states the game could be in, in terms of game modes and windows open.

Testing the threading was usually difficult, as bugs crept up seemingly randomly. The timing of user key inputs seemed to create some threading problems under our old system. We tested the system by pressing the keys profusely.

3.2 Results

3.2.1 Physics

1) Unit Testing

The physics engine passes unit tests in most cases, but it is unclear what the correct output should be for some edge cases. For fast movement of the ball, it was found that the approximate collision detector was not always accurate enough to detect the collision. In addition, when the ball and a lateral cylinder intersect, the output of the collision methods was found to be not consistent and dependent on some hidden factors.

2) checkRep()

The physics package runs without violating assertions for many runs of the physics engine in the widely varied applications of Gizmo Arcade. This gives a rather high degree of confidence that each state of the physics package is internally consistent and should eliminate bugs involving attributes of PhysicsModels such as bounding sphere.

3) Stress Testing

The physics engine runs without deviation from realistic physics in most cases; however, there are several special cases where strange effects happen. The success of our engine in dealing with most other cases implies that we can have a relatively high level of confidence in its correctness, since errors occur only in strange cases that the physics engine was arguably

not designed to handle.

3.2.2 Graphical User Interface

1 Dialog
2 GamePlay
3 GamePlayModel
4 PaletteModel
5 PlayScreen
6 Screen
7 PaletteScreen
8 UserAction
9 WelcomeScreen
10 WorldModel

1 Dialog
Black box required functionality
1 for loading and saving, the files are filtered so that the user is defaulted to only see .XML files
2 for loading and saving, there is the *.* file filter option as well, in case the user wants to risk breaking the contract
3 closing the dialog does not alter game objects
4 for loading and saving, user can see directories
5 for loading and saving, user can select an XML file and see it loaded or saved
6 invoking a dialog pauses the game if the game is in play mode
7 canceling out of a dialog reverts the game mode to its original state
8 message dialogs show up correctly, pauses the game if game is not paused, and returns game to previous state if dialog is exited
9 pressing game control keys while the window is displayed should not work
10 the game window itself should not be focusable while the window is displayed

Testing
1 the window was invoked and I checked to see that the default filter was *.XML files
2 the *.* file filter option was existent
3 closing the window did nothing to the state of the world
4 I could see directories
5 I could select an XML file among multiple ones, and the correct one would load
6, 7 to test these aspects, I tried invoking the screen from all possible game play modes, and verified the desired responses

8 required functionality verified
9 game control keys did not register while the window was displayed
10 the game window was not focusable while the window was displayed

Glass box required functionality
1 for loading and saving, JFileChooser.APPROVE_OPTION is toggled whenever a user chooses a desired file

Testing
1 Required functionality verified

2 GamePlay
Black box required functionality
1 Welcome screen should show up
2 GamePlayModel and PaletteModel must be correctly loaded

Testing
1 The window shows up without too much delay, correctly
2 Required functionality verified

3 GamePlayModel
Black box required functionality
1 correctly initializes
2 correctly runs the game at rate framerate
3 there is at most one thread that runs a game on a world at a given time
4 keyboard input correctly invokes the functions they are linked to
5 stalling functions, such as quit(), load(), or pressing escape to go back to the welcome screen, which requires a dialog box to pop up and the user to respond, always pauses the game first if the game is in play mode; afterwards, the game mode is reverted to what it was before the dialog box pop up. In particular, if the original state was in pause mode, no changes would occur

Testing
1 required functionality verified
2 checked via observing intervals of update
3 checked via asking for Thread.activeCount() and checking whether the count increases every time I load a game world
4 tested via pressing keys that were predefined

5 tested via trying all possible starting game modes

4 PaletteModel

Black box required functionality

1 clicking various components that show up to the user should be reflected as changes in the internal model

2 currentGizmo should correspond to the current Gizmo that is focused

3 properties of currentGizmo should correspond to what is shown to the user via observing windows

4 saving changes should update the current world

5 not saving changes should revert to the old world

6 game control keys should not work in edit mode

7 functions in PaletteModel should only be enabled during edit mode

5 PlayScreen

Black box required functionality

1 the window should show the game being played

2 the user should be able to see all of the gizmos in the world state

3 the size of the screen should be 600x600 (subject to change)

4 menubar should have the following structure (-- is a separator)

- File
    - New Game
    - --
    - Save Game
    - Save Game As…
    - --
    - Load Game
    - --
    - Return to Main Menu
    - Quit Game
- Game Settings
    - Pause Game
    - Hide Wall Wireframe (Show Wall Wireframe)
    - Go to Edit Mode
- Camera Settings
    - Shift Camera Right
    - Shift Camera Left
    - Shift Camera Up

- Shift Camera Down
- Reset Camera

- Help
  - Hot keys…

5 New Game should ask the user to choose from a world
6 Save Game should open up a JFileChooser if there is no current file being worked on; otherwise, no JFileChooser should open and the current game being played should be saved onto the file
7 in editing mode, clicking on a gizmo should focus the current gizmo to be that gizmo
8 in editing mode, clicking on a void when create gizmo is toggled should create that gizmo there
9 in editing mode, clicking on a gizmo when create gizmo is toggled should untoggle creation and focus on that gizmo
10 in editing mode, exiting editing mode should relinquish focus on any gizmo and untoggle the creation of gizmo
11 in editing mode, saving should exit edit mode first
12 during play mode, no selected-ness graphics should remain from editing mode (during editing mode, clicking on a gizmo gives it selected-ness graphics)

Testing
1 tested
2 tested
3 tested
4 tested
5 tested
6 tested
7 tested
8 tested
9 tested
10 tested
11 tested
12 tested, but sometimes found problems

6 Screen
Black box required functionality
1 abstract class should implement the key inputs correctly

Testing
1 tested

7 PaletteScreen
Black box required functionality
1 panels display as desired
2 undo correctly undoes last action
3 redo works
4 done/undone action items show up correctly in the history list
5 clicking on the history list performs desired actions (specified earlier)
6 clicking on a Gizmo to create makes the button have a yellow background with blue border
7 menu items work properly
8 user is able exit by pressing menu or button
9 menubar should have the following structure (-- is a separator)

- Edit
    - Undo
    - Redo
    - Check for Gizmo overlap
    - Refresh
    - --
    - Exit

Testing
1 tested
2 tested with respect to each action. All of the types of actions were tested as well as variations for each action.
3 tested as with undo
4 tested
5 tested
6 tested
7 tested
8 tested
9 tested

Glass box required functionality
1 Scroll pane's viewable bounds correctly update as one undoes/redoes actions

Testing

1 tested

8 KeyableAction
Black box required functionality
1 the set of Gizmos specified should be activated upon the invocation of the KeyableAction

Testing
1 testing done for implemented classes

9 WelcomeScreen
Black box required functionality
1 user should see three buttons – Start New Game, Load Saved Game, and Quit
2 Start New Game should load a default world and close the window
3 Load Saved Game should bring up a JFileChooser so that the user can load an XML file

Testing
1 tested
2 tested
3 tested

10 WorldModel
Black box required functionality
1 public method allows XMLParser to add gizmos and triggers to the WorldModel
2 calling setDefaultFlipperKeys() correctly initializes keys to the gizmos they are supposed to activate

Testing
1 XMLParser was used to extensively test the loading scheme
2 Required functionality verified

IV. Reflection

4.1. Graphical User Interface

Writing the GUI, as expected, took a lot of brute-force tedious work, especially with the menu items, buttons, and text fields. There was a fair amount of thinking involved, however, in getting the game together by putting together all the pieces. I was happy to know about other

parts of the project, although it was only a titular recognition. I knew all about my GUI but not much about anything else.

I had fun creating the GUI and making it as intuitive as possible. Although I created many of the features because I thought they were intuitive, I am hoping that they are intuitive for everyone.

4.2 Evaluation

4.2.1 Graphical User Interface

I believe the GUI turned out to implement most of the things I wanted. If there was a burning feature I wanted, I usually ended up sitting down and chugging through until it was completed. This led to a lot of time being spent on my part that is technically unnecessary since most of my features were not required.

Because of time constraints, I was not able to implement everything that I wanted to implement for the GUI. My goal would be to make the interface more intuitive. Brand new features are nice, but my main purpose in writing the GUI was to make everything as intuitive as possible.

4.2.2 Physics

The physics system developed has a great deal of flexibility and is able to achieve a great deal of functionality from a relatively small amount of basic source. The use of primitive PhysicsObjects with common functionality and PhysicsModels that use the functionality of the contained PhysicsObjects allows for desired effects to be implemented once in each PhysicsObject and thus extended to all aspects of physics.

4.3 Lessons

4.3.1 Graphical User Interface

I could have saved a lot of time by planning more ahead of time. The module dependency diagram should have been thought out more, since I was sometimes confused about my modules.

4.3.2. XML Parser

While the XML Parser ended up satisfying all the requirements in the end, it did not include full support for all of GB Arcade's features. This was partly due to the fact that certain aspects of

the parser were not well thought out initially, and forced time to be spent on revision that could have otherwise been spent improving the parser.

4.4 Known Bugs and Limitations

4.4.1 Editor

There is a known bug in the editor. It is not easily reproduced. When a Gizmo is clicked in the editing mode, it is painted by a certain graphic that allows the user to know that it has been clicked. Upon exiting the editing mode, that graphic should disappear and the original graphic should display. However, sometimes, that graphic does not disappear.

There is also a limitation in the editor, in the sense that future developers will have a hard time adding more user input TextFields. The program was made without much design thought to the input space, with the layout mostly taken from NetBeans, although, if the user uses NetBeans, it would be fine.

There are also a couple of minor annoyances in the editor. If the user disallows overlap, he/she cannot edit certain properties of a Gizmo if it is in an illegal spot unless it is a key mapping edit. For example, intuitively, one should only be unable to update an illegally placed Gizmo only if the position is being edited. Things like texture should still be editable. Also, the scroll pane for the history of actions has to scroll appropriately so that the most recently done action element is at the top of the scroll pane. Sometimes, this is not the case, and the scroll pane is off by a couple pixels.

4.4.2 Physics

When the coefficient of reflection of a flipper is significantly less than 1.0, the ball will sometimes sink into the flipper, causing strange effects. In addition, when on top of a rotating cylinder with coefficient of reflection less than 1.0, the ball will sometimes actually sink into the cylinder. We believe that these problems are largely due to the fact that the physics engine was not built to deal with objects sitting on top of a surface.