Stephanie Keck
UIN 526009919
CSCE 312 506
30 March 2018

Project X Game Proposal: Breakout with Novelty Elements

I. Game Objective

I intend to build a version of the arcade game Breakout with unconventional elements taken from other games such as pinball. Users are presented with a ball, a paddle that can move horizontally at the bottom of the screen, and a number of blocks which disappear when they are hit by the ball. The objective of the game is to clear all blocks from the screen without allowing the ball to become lost by falling below the paddle, while also accumulating the maximum amount of points.

II. Game Rules

The following rules will govern gameplay:

Paddle — Keyboard input is used to move the paddle horizontally. The paddle is used to hit the ball back into the game field above.

Ball — The player gets a predetermined number of balls; losing a ball by letting it fall below the paddle decreases this count. When the player has lost all balls, the game is lost. Catching randomly generated drops with the paddle may increase or decrease the remaining number of balls.

Blocks — A number of blocks are present in the gameplay field. Blocks are visually distinct from other gameplay elements. The goal of the game is to clear all blocks from the field. Blocks are typically cleared by hitting them once with the ball. However, some blocks may take several hits to clear. The ball may also have to interact with other gameplay elements before some blocks can be cleared.

Other elements — Other elements on the game field do not need to be cleared but may have various effects on gameplay. Keyboard-controlled fins like those found in pinball, for instance, may give players another way to control the trajectory of the ball.

Drops — Drops are objects that the player can “catch” with their paddle that are generated by clearing blocks or interacting with other game elements. They may have either beneficial or negative effects on gameplay. One drop may grant an extra ball while another may shrink the paddle.

Points — Everything a player does in the game can generate points. Clearing blocks will typically add a fixed number of points. Interacting with other game elements may also earn points. Points earned on all levels will be displayed at the end of each game.

Levels — The game starts at level one and progresses through a fixed number of levels. Each level has different objects and challenges. Players must win all levels to beat the game.

III. Implementation Overview

The following is a description of the preliminary details of the implementation of this application. Note that this is not exhaustive and omits some low-level details; however, it should give a good enough idea of how the major components of this game will work.

The `MainController` class represents the topmost level in the application hierarchy below `Main`. It controls the main game loop, handles user input, and delegates other tasks to its internal `GameController`. It contains the following methods:

> `method void start()` — Initializes the main application elements and starts the main loop. While the loop is running, the `MainController` continually calls `update()` on its GameController when the application is not paused. This drives the behavior of the application. This method is called by the `Main` class as soon as the program starts.
>
> `method void pause()` — Sets an internal variable, causing the main loop to stop calling `update()` on the `GameController`. This has the effect of freezing the game.
>
> `method void unpause()` — Undoes the effects of calling `pause()`.
>
> `method void quit()` — Stops the main loop and deallocates all memory for the application by calling `destroy()`.
>
> `method void destroy()` — Deallocates all memory for this class and its fields.

The `GameController` handles high-level gameplay logic, namely levels and displaying player stats. It delegates all other tasks to one object in its list of `LevelControllers`; that object represents the current level. It contains the following methods:

> `method void update()` — This function represents one "tick" of the main loop. Here `GameController` queries its current `LevelController` to retrieve relevant player stats and determine if it should advance the game to the next level.
>
> `method void onKey(int key)` — Handles keyboard input passed by `MainController`; `MainController` will pass `GameController` any input it does not know what to do with.
>
> `method void reset()` — Resets the game to initial conditions.
>
> `method void destroy()` — Deallocates all memory for this class and its fields.

`method void startLevel()` — Starts the game for the current level (i.e., the ball starts moving) and updates appropriate UI.

`method void nextLevel()` — Stops the current `LevelController` and changes an internal variable so that the current `LevelController` is now the next one in the list. Updates the appropriate UI.

`method void updateStatsUI()` — Retrieves stats from the `LevelController` and updates appropriate UI.

`method void showBallLostUI()` — Shown when the player loses a ball but still has balls remaining. This UI will display a "try again" message until the player presses a key, at which point the level will continue with the position of the ball reset. This behavior is handled in the `update()` method.

`method void showGameOverUI()` — Shown when the player loses all balls. When the player presses a key, the game starts over at level one with all elements reset to initial conditions. This behavior is handled in the `update()` method.

`method void showPlayerWonUI()` — Shown when the player clears all blocks on all levels. When the player presses a key, the game starts over at level one with all elements reset to initial conditions. This behavior is handled in the `update()` method.

The primary purpose of the `LevelController` class is to mediate between the `GameController` and its internal `FieldController`; it keeps track of player stats and outcomes while the `FieldController` handles actual game object interactions. Its methods include:

`method void update()` — This function represents one "tick" of the main loop. The `FieldController` is queried for points generated this tick, blocks remaining, whether the ball was lost, and other relevant stats.

`method void onKey(int key)` — Handles keyboard input passed by `GameController`.

`method void destroy()` — Deallocates all memory for this class and its fields.

`method void startLevel()` — Starts the game for the current level (i.e., the ball starts moving).

`method void resetBall()` — Calls `FieldController.resetBall()`, which resets the ball to its initial position.

`method void handleDrop(String drop)` — Handles a drop passed by the `FieldController`; effects may include increasing or decreasing ball count, increasing points, etc. depending on the type of drop.

`method void showUI()` — Shows the UI for this level, including calling `FieldController.showUI()`. This method is needed so that all levels contained in `GameController` do not display at once.

`method void hideUI()` — Hides the UI for this level, including calling `FieldController.hideUI()`. This method is needed so that all levels contained in `GameController` do not display at once.

`method int getPointCount()` — Returns the current player point count for this level.

`method int getBlockCount()` — Returns the remaining block count for this level.

`method int getBallsRemaining()` — Returns the amount of balls remaining for this level.

`method bool levelWon()` — Called by `GamerController` when the level is lost or won.

`method bool ballLost()` — Returns whether the ball was lost this tick.

The `FieldController` handles much of the low-level game logic. It is responsible for detecting collision between objects, generating random drops, and calculating the trajectory of the ball, among other things. The controller also contains all of the visible game objects associated with the game field, including the ball, paddle, and blocks. These methods address the highest-level functionality of this class:

`method void update()` — This function represents one "tick" of the main loop. Here game objects are tracked and updated, collisions are checked, drops are generated, etc.

`method void onKey(int key)` — Handles keyboard input passed by `LevelController`. Depending on the key, may call `Paddle.moveLeft()` or `Paddle.moveRight()`.

`method void destroy()` — Deallocates all memory for this class and its fields.

`method void startBall()` — Starts the ball moving.

`method void resetBall()` — Resets the ball to its initial position.

`method void showUI()` — Shows the UI for this game field. This method is needed so that all levels contained in `GameController` do not display at once.

`method void hideUI()` — Hides the UI for this game field. This method is needed so that all levels contained in `GameController` do not display at once.

`method void add<Object>(<Parameters>)` — This class of methods is how a game field is configured for a level. A method of this signature can be called for every type of game object besides the ball, paddle, or drops, in order to add it to the field. Relevant parameters will configure the object; these may include position, size, color, and other features. For now configuration of fields will be hardcoded into the constructor of the `LevelController`; however, I may implement a more flexible way to configure levels if time permits.

`method int getPointCount()` — Returns points generated by object interactions this tick.

`method int getBlocksRemaining()` — Returns the count of blocks remaining in this game field.

`method bool ballLost()` — Returns whether the ball was lost this tick.

`method void handleDrop(String drop)` — Handles any drops generated by `update()`. Any drops not handled by this method are returned by `getDrop()` to be handled by `LevelController`.

`method bool dropAcquired()` — Returns whether the player acquired a drop (typically by catching it with the paddle). `LevelController` will call this to determine if it needs to retrieve the drop via `getDrop()`.

`method String getDrop()` — Returns a drop acquired this tick.

Finally, the various game objects each also have associated controllers. The details of some of them are listed below. During development I will focus on adding various novel and interesting objects to the game field; one idea is to implement keyboard-controlled fins and other elements typically found in pinball. The following controllers are the most important to consider for now:

`BallController`:

`method void update()` — This function represents one "tick" of the main loop. The ball will continually move towards the point last set via its `setTarget()` method after it has been put in motion.

`method void destroy()` — Deallocates all memory for this class and its fields.

`method void showUI()` — Shows the UI for this object. This method is needed so that all levels contained in `GameController` do not display at once. Note that all objects which can be present in a game field will have this method.

`method void hideUI()` — Hides the UI for this object. This method is needed so that all levels contained in `GameController` do not display at once. Note that all objects which can be present in a game field will have this method.

`method void start()` — Starts the ball moving toward the point last set via its `setTarget()` method.

`method void stop()` — Stops the ball.

`method void setTarget(int x, int y)` — Sets a point on the screen that the ball will continually move towards while it is in motion.

`method void setSpeed(int speed)` —Sets the speed of the ball, or how many pixels it will move per tick.

`method int getXPosition()` — Returns the x-position of this object.

`method int getYPosition()` — Returns the y-position of this object.

`PaddleController`:

`method void destroy()` — Deallocates all memory for this class and its fields.

`method void showUI()` — Shows the UI for this object. This method is needed so that all levels contained in `GameController` do not display at once. Note that all objects which can be present in a game field will have this method.

`method void hideUI()` — Hides the UI for this object. This method is needed so that all levels contained in `GameController` do not display at once. Note that all objects which can be present in a game field will have this method.

`method void unfreeze()` — Allow this object to move in response to `moveLeft()` and `moveRight()`.

`method void freeze()` — Disallows this object from moving in response to `moveLeft()` and `moveRight()`.

`method void moveLeft()` — Moves the paddle left; distance depends on the set speed.

`method void moveRight()` — Moves the paddle right; distance depends on the set speed.

`method void setSpeed(int speed)` — Determines how many pixels the paddle will move in a single call to `moveLeft()` or `moveRight()`.

`method int getSpeed()` — Returns this paddle's speed setting.

`method int getPosition()` — Returns the horizontal position of this object.

`method int getSize()` — Returns the width of this object.

`method void setPosition(int x)` — Sets the horizontal position of this object.

`method void setSize()` — Sets the width of this object; drops may grow or shrink the paddle.

`BlockController`:

`method void destroy()` — Deallocates all memory for this class and its fields.

`method void showUI()` — Shows the UI for this object. This method is needed so that all levels contained in `GameController` do not display at once. Note that all objects which can be present in a game field will have this method.

`method void hideUI()` — Hides the UI for this object. This method is needed so that all levels contained in `GameController` do not display at once. Note that all objects which can be present in a game field will have this method.

`method int getXPosition()` — Returns the x-position of this object.

`method int getYPosition()` — Returns the y-position of this object.

`method void clear()` — Sets the status of this object to cleared; used by FieldController.

`method bool isCleared()` — Returns whether this object has been cleared; used by FieldController.

Classes planned but not covered here include `DropController` and controllers for other game objects. These will be very similar to the game object controllers mentioned above.