Walkthrough

Group_0633: Lance Oribello, Neel Patel, Ishan Sharma, Nancy Zhao

Presentation Outline

- **Introduction** of the team and each member's various contributions to the project in accordance with TEAM.md.
- Explain the features and descriptions for both new games: **Snake and Blocks**.
- Click through **game centre** to show:
    - Account sign up and logins.
    - The Snake and Blocks games.
    - Show how the games interact with the scoreboard. Showcase high scores from multiple accounts.
- Run through **testing**:
    - Present the 100% unittest class and how it was developed.
    - Present the 0% unittest class and why we decided not to write tests for it.
- Present rough UML diagram of our overall class hierarchy.
- Explain how we utilized the **MVC design pattern** in the following classes:
    - The logic classes of Snake and Blocks are separated from their view classes.
    - Clarify how we combined model and controller in those examples.
- Delve more into Blocks design:
    - Logic of Grid class and how it interacts with GridManager.
    - BlocksView and how things are displayed.
- Explain how the scoring system is implemented and designed.
- Activity and controller classes for lobby: Login, Game Select, Scoreboard.
    - How they were split and why we have done so.
- Explain how we used the **Memento design pattern**:
    - In saving Boards and Grids in an arraylist in their respective managers.
    - In saving a Snake game's data into an object array.

Unit Test Coverage

What is your unit test coverage?

Logic/Controller Classes

- LoginController: 100% lines covered
- GameSelectController: 100% lines covered
- ScoreboardController: 100% lines covered
- UserAccount: 100% lines covered
- SlidingTilesMenuController: 100% lines covered
- Board: 100% lines covered
- BoardManager: 100% lines covered
- Tile: 100% lines covered

- SnakeMenuController: 100% lines covered
- SnakeController: 100% lines covered
- BlocksMenuController: 100% lines covered
- Grid: 100% lines covered
- GridManager: 100% lines covered

Display/View Classes

- Separated classes in order to test logic separately from the activity/view, so all classes concerning display were not tested.

Classes

What are the most important classes in your program?

Lobby Classes

- LoginController
    - The control of the login screen shown on app startup.
    - Processes sign ups and logins of user accounts by referencing user account list.
- ScoreboardActivity
    - The display of the per-user and per-game scoreboards of the current user account shown after clicking the view scoreboards button in game select screen.
    - Creates and displays the table of scores where the columns are the score types and the rows are the scores of a particular game level.
- ScoreboardController
    - The control of the scoreboard screen.
    - Processes the top scorers of each game level and their scores into a string array and the top scores of each game level for the current user into a string array for display in the activity.
- UserAccount
    - Stores a user account's information: username, password, top scores of each game level, and game names for each game.
    - Holds a class variable of the names of the game levels.

Sliding Tiles Classes

- BoardManager
    - Manages a board, including initializing a solvable game, swapping tiles, undo, checking for a win, and managing taps.
    - Keeps track of the complexity chosen, the number of moves made, and the boards corresponding to each move.

Snake Classes

- SnakeController
    - The controller for Snake, manages the Snake logic.
    - Encompasses the starting and resuming of games, the saving and loading of games, setting the difficulty of the game, spawning bombs and apples, and managing the snake's growth and movement.

Blocks Classes

- BlocksView
    - The view for Blocks, displays the Blocks game on the screen.
    - Contains the directional input buttons and the displayed game grid.
    - Processes movement and block placement taps.
- Grid
    - The grid used for the Blocks game.
    - A new grid can be initialized to begin a new game, or a grid from a previous game can be loaded.
    - Contains methods for changing the grid's values, including moving the player, spawning food, placing blocks.
- GridManager
    - Manages a grid, includes placing blocks, moving the player, and undoing turns.
    - Keeps track of the moves made by saving the grids corresponding to each move.

Design Patterns

What design patterns did you use? What problems do each of them solve?

Model View Controller

Implementation

- The various activities of the repository have their own respective controllers.
- The logic classes of Snake and Blocks are separated from their respective view classes.

Problem Solved

- Separated logic from view to optimize testing.
- Gives each class a more clear purpose.
- Organizes the classes more elegantly.

Memento

Implementation

- Grids and Boards are saved into arraylists in their respective managers.
- Snake saveData is written into the file.

Problem Solved

- Made it easier to implement an unlimited undo move functionality in Sliding Tiles and Blocks.
- Made it easier to load past game data to resume games.

Iterator

Implementation

- Board uses it to traverse through it's Tiles.

Problem Solved

- The tiles are set as a 2 dimensional array of integers. Traversing through a 2-dimensional array naturally would require a nested loop every time an individual wishes to iterate through the tiles.
- Implementing an iterator solves this issue as it lets the user traverse through our 2 dimensional array of Tiles in a more simple manner which would only normally consist of a single loop every time the iterator is used, instead of needing to create a nested loop.

Builder

Implementation

- Grids can be created either from scratch or using a constructor that systematically constructs Grid from given data for the purposes of loading past Grids.

Problem Solved

- A Grid is now easily constructed from easy to obtain data, optimizing how different representations of a Grid can be created.

Scoreboard

How did you design your scoreboard? Where are high scores stored? How do they get displayed?

Design

- The View Scoreboards button is located on the Game Select Screen, which appears after the user has logged in.
- Upon clicking the button, the **per-user scoreboard** (top scoring user of each game level and their score) and the **per-game scoreboard** (top score of current user for each game level) are displayed. The game levels are:
  - Sliding Tiles 3x3
  - Sliding Tiles 4x4
  - Sliding Tiles 5x5
  - Snake Easy Mode
  - Snake Hard Mode
  - Blocks
- The scoreboards are updated every time a new top score is achieved in each of the three games, so when the user returns to the Scoreboard Screen, the new top scores are reflected on the scoreboards, if applicable.
- Scoreboard is designed using the **Model View Controller** (MVC) design pattern:
  - Model: the score data is stored in the user accounts, which is stored in the user account list file (**accounts.ser**) which allows for data to be loaded upon reopening the app and logging in from different accounts.
  - View: the display is created through **ScoreboardActivity**, which displays the layout and its TextView and TableLayout, and interacts with the user account list file to display the top scorers and top scores.
  - Controller: the top scoring users and top scores of the current user are retrieved through **ScoreboardController**, which contains the logic for getting the scores into a String array for display.

Storage

- The top scores of a user are stored in the UserAccount object in the instance variable **scores**, which is a Map of Strings, the game level name, to Integer, the top score of the user of that game level.
- The game levels are stored in a string array class variable in UserAccount **gameLevels**, so if a new game level is added, only that variable needs to be modified in UserAccount and not anywhere else like in ScoreboardController.

- The Map scores is private in UserAccount, and top scores are accessed in scoreboard (to display the top score) and in the games (for determining when to update the top score) by using public method **getTopScore**, which takes in a game level name and returns the top score of the user for that game level.
- For a new user, or a user who has not played the game level yet and hence has not had the top score for that game level updated, the default base score is 1000000 for Sliding Tiles (where the goal is to get as low a score as possible) and 0 for Snake and Board (where the goal is to get as high a score as possible). These default scores do not appear on the Scoreboard Screen, and instead display as “None”, which is checked in ScoreboardController. The purpose of these default scores is to have a basis for comparison when new top scores are added, and to be able to show “None” when the user has no top score for the game currently.
- To set a top score, public method **setTopScore** can be called, which takes in a game level name String and a new score Integer, and replaces the current top score in the Map with the one passed in. By using replace, null is returned if this method is used with a non-existing game level name passed.
- UserAccount has an equals method which also checks if the top scores of the users compared match.
- All user accounts are saved in an ArrayList of UserAccount (which implements Serializable), which is saved to **accounts.ser**. This file is updated every time a top score is updated in each of the games, and is updated along with the current user account in GameSelectActivity and ScoreboardActivity (in their onResume methods) so that the scoreboards are updated when the user returns to those screens. The current user account object is also passed along the activities using Intent, so the games can update the top scores of the correct user account.

Display

- The two scoreboards are displayed in a **TableLayout**, where the rows are the game levels and the columns are the two score types: per-user and per-game.
- The scoreboard title is first generated with **TextView** which is customized for the current user who is logged in. This is done in the method **createScoreboardTitle**.
- The scoreboards are generated with a TableLayout in the method **createScoreboardTable**, which calls on the Controller class’s logic methods to get the top scorers and top scores in array form.
- Then it iterates through the game levels (stored as class variable gameLevels, a String array in UserAccount), adding a heading TextView for each first to the TableRow, and filling their respective score columns with the top scoring user and top score of the current user by creating a TextView for each and adding it to the TableRow. At the end of each loop, the **TableRow** with three columns (heading, per-user, per-game) is added to the TableLayout.

Using the Application

A comprehensive description of our app's features and instructions for how to use it (note, our presentation will run through our app's features, but not necessarily in this much detail; a copy of this application "appendix" is included within TEAM.md):

Login Screen

- The first screen of the application prompts for a username and password. Here you may sign up a new user account to keep track of your scores and saved games.
- Input your desired username and password and then press signup; a toast is shown whether the sign up is successful. Provided that an account with that username does not already exist, you may now press the login button to continue to the game select screen.

Game Select Screen

- Here you can find buttons that lead you to the menu screens of our three games: Sliding Tiles, Snake, and our original game Blocks. There is also a button here that allows you to view the scoreboard of the games.

Scoreboard Screen

- The scoreboard displays the top-scoring user and their score for any one of the various modes of our games (under Per-User), as well as the current user's top scores for each of those game modes (under Per-Game).

Sliding Tiles

- Pressing the Sliding Tiles button in the Game Select Screen leads you to the Sliding Tiles menu. Here you can find buttons for loading and saving the game, loading an auto save, undoing the moves of the current game, and starting a new game.
- Pressing the new game button prompts you to choose from three complexities: 3x3, 4x4, and 5x5, each corresponding to a differently-sized game board with a different number of tiles.
- The game itself is simple: the board contains numerous tiles, each with a number on them except for one blank tile. To win the game, you must slide the tiles into row-major order with the numbers ascending and the blank tile at the bottom right. You swap tiles by pressing any of the tiles adjacent to the blank tile, which results in that tile switching with the blank tile. A single tile swap constitutes one move. Your final score is determined by the number of moves you take to win: the lower the better! Your high score is updated upon finishing the game if you beat your old one.

- Pressing the back button gives you access to the menu once again, where you can save the game in progress or undo a number of moves (you input how many you wish to undo).
- When games are saved manually, they are labelled with the exact time and date of saving.
- If you have any past saved games, pressing load game will bring up a list of all your previous saves; clicking on any of them automatically brings up that game to play. An autoSave is automatically created in your account whenever you exit the game screen; pressing the Load Auto Save button in the menu loads that autoSave game back up.

Snake

- Pressing the Snake button in the Game Select Screen leads you to the Snake game menu. Here you can find buttons for loading and saving the game, loading an auto save, and starting a new game.
- Pressing the new game button prompts you to select a difficulty level: easy or hard, the difference in the two being in how fast your snake moves.
- The game itself is very much like classic Snake: maneuver the snake around the screen eating apples, which cause the snake to grow in length and your score to increase.
- Also present in the game are bombs, which end the game when eaten. Every time an apple is eaten, a new apple is spawned in a random location, and the bomb's location is randomly changed.
- Every three apples eaten, the game creates an automatic save point, which can be loaded from the menu. The game ends when a bomb is eaten, the snake runs into the border of the play area, or the snake eats one of its own body segments.
- Whenever the snake reaches a certain length, the game's difficulty increases! Your snake's length is reset to 1 and the game runs a little bit faster.
- The snake is controlled with four buttons at the bottom of the screen, corresponding to up, down, left, and right. Note you cannot change your direction to the direction opposite to the one you are currently moving in (lest you turn back into yourself!).
- A pause button is also present at the bottom right of the screen, which allows you to stop the snake's movement until you press it again.
- Pressing the back button gives you access to the menu once again, where you can save the game in progress, load any old saved games (the Load Game button also gives you access to any save points created), or load an auto save.
- When games are saved manually, they are labelled with the exact time and date of saving.
- Similarly to Sliding Tiles, an autoSave is automatically created in your account whenever you exit the game screen; pressing the Load Auto Save button in the menu loads that autoSave game back up.
- Your high score is also updated whenever you exit the game screen if you beat your old one.

Blocks

- Pressing the Blocks button in the Game Select Screen leads you to the Blocks game menu. Here you can find buttons for loading and saving the game, loading an auto save, undoing the moves of the current game, and starting a new game.
- Pressing the new game button begins a new game of Blocks, a game of our original creation.
- The game is played on a 9x9 grid which are populated by blocks, food, and the player.
- The borders of the grid are taken up by blocks. Starting a new game places the player in the middle of the grid and spawns 4 foods in random locations around the grid.
- The aim of the game is to collect as much food as you can until your movement is completely hindered by (possibly your own) blocks, upon which it is game over!
- A turn in this game constitutes doing one of two things: moving the player or placing a block.
- When the player is moved, it must move the maximum number of squares in the up, down, left, or right direction until it encounters a food or a block. A player stops at the food's location and eats it, and the player stops before blocks. Eating food causes another food to spawn at random location; there is always 4 food on the screen at a time.
- Movement is controlled using the directional input buttons at the bottom of the screen.
- The player places blocks by tapping the screen; blocks may only be placed in empty squares.
- Blocks are used to purposefully impede your own movement to obtain any food that is out of your reach.
- As with Sliding Tiles, pressing the back button gives you access to the menu once again, where you can save the game in progress or undo a number of moves (you input how many you wish to undo).
- When games are saved manually, they are labelled with the exact time and date of saving.
- If you have any past saved games, pressing load game will bring up a list of all your previous saves; clicking on any of them automatically brings up that game to play. An autoSave is automatically created in your account whenever you exit the game screen; pressing the Load Auto Save button in the menu loads that autoSave game back up.
- Your high score is also updated whenever you exit the game screen if you beat your old one.

Repository Link: https://markus.teach.cs.toronto.edu/git/csc207-2018-09-reg/group_0633