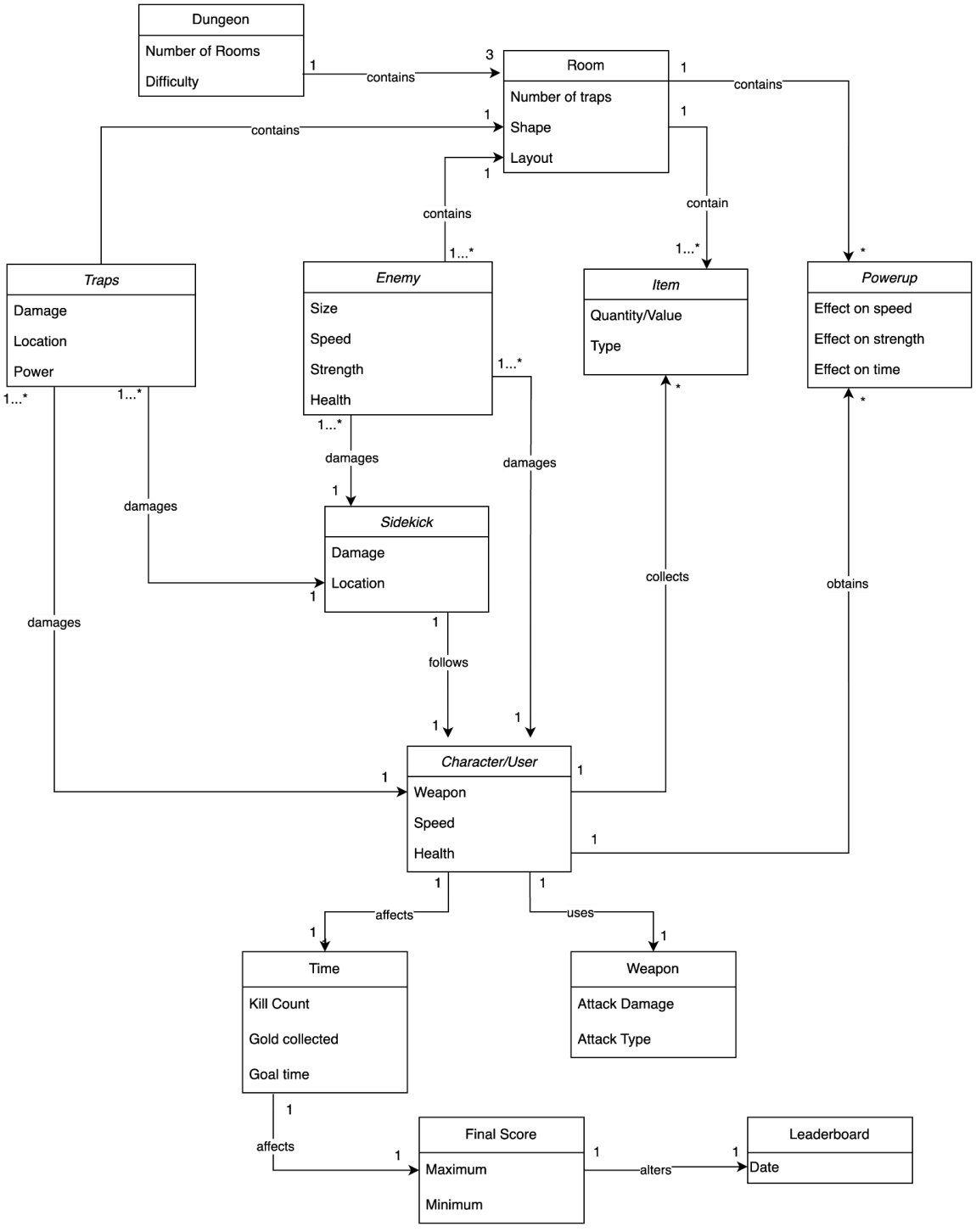
Revised Domain Model

1. Nouns
    a. Dungeon
    b. Player
    c. Layouts
    d. Enemy
    e. Attack
    f. Powerups
    g. Exits
    h. Rooms
    i. Health
    j. Score
    k. Leaderboard
    l. Final Score
2. Classes + Attributes
    a. Dungeon (Class)
        i. Number of Rooms (Attribute of Dungeon)
        ii. Difficulty (Attribute of Dungeon)
    b. Room
        i. Trap
        ii. Shape
        iii. Layout
    c. Trap
        i. Damage
        ii. Location
        iii. Power
    d. Enemy
        i. Size
        ii. Speed
        iii. Strength
        iv. Health
    e. Item
        i. Gold
        ii. Special Collectable
        iii. Quantity/Value
    f. Power-up
        i. Increased Speed
        ii. Increased Strength
        iii. Time Decrease
    g. Sidekick
        i. Attack Damage
        ii. Health
    h. Character
        i. Weapon

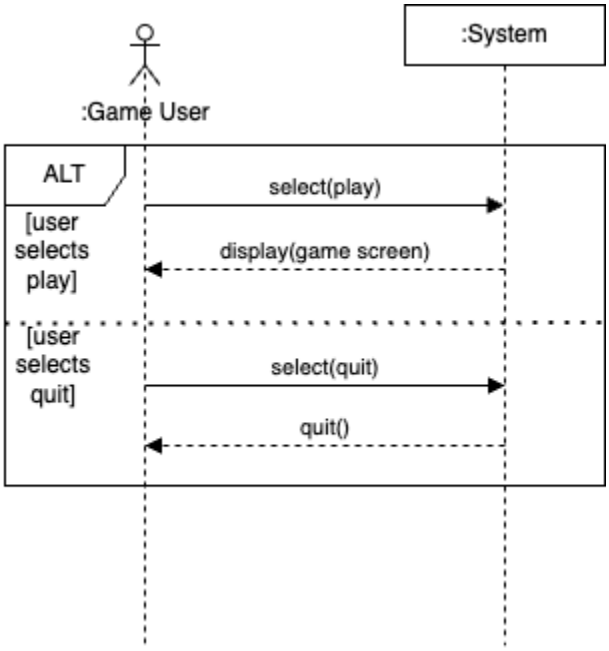
    ii. Speed
    iii. Health
i. Weapon
    i. Durability
    ii. Attack Damage
    iii. Attack Type
j. Task
    i. Target enemy count
    ii. Target gold count
    iii. Time limit
k. Time
    i. Kill count
    ii. Gold collected
    iii. Goal time
l. Leaderboard
    i. Date
m. Final Score
    i. Maximum
    ii. Minimum

3. Domain Model

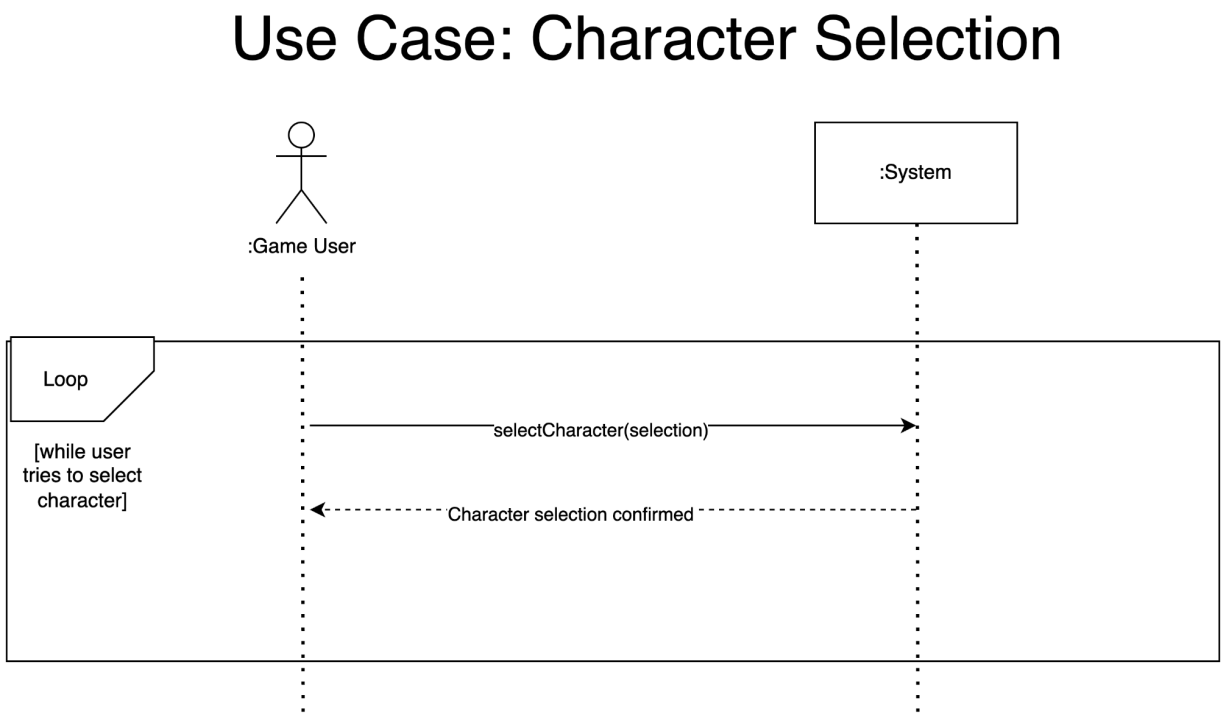


System Sequence Diagrams

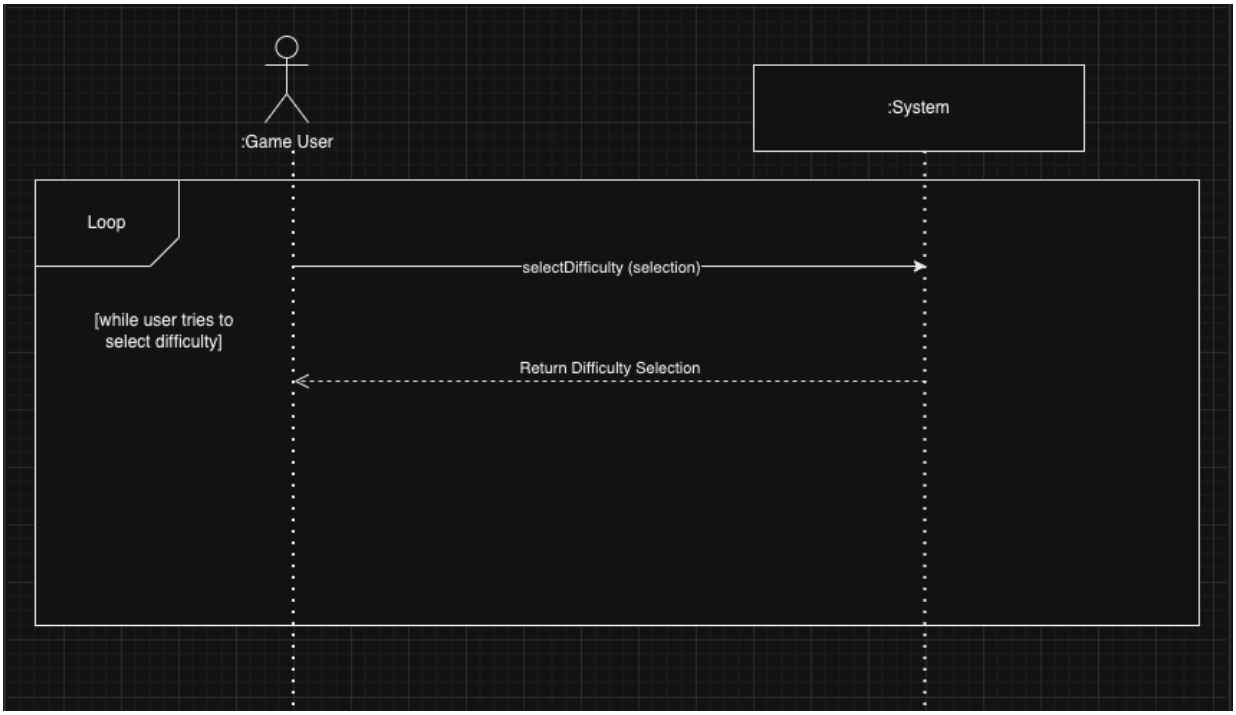
1. Play Game - Kavya



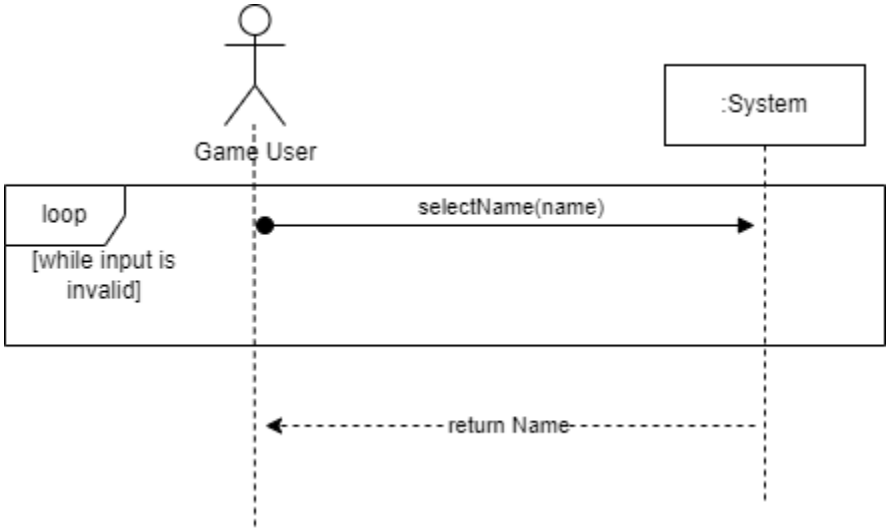
2. Pick Character - Daniel



3. Pick difficulty - Rashmith



4. Select Name - Caleb



Design Pattern Evidence

```
7 ages  Caljorb
public class LeaderboardViewModel extends ViewModel {
    10 usages
    private ArrayList<LeaderboardRow> leaderboardRows;
    3 usages
    private static LeaderboardViewModel leaderboardViewModel;

    1 usage  Caljorb
    private LeaderboardViewModel() {
        leaderboardRows = new ArrayList<>( initialCapacity: 5);
        for (int i = 0; i < 5; i++) {
            leaderboardRows.add(new LeaderboardRow( name: "",  score: 0,  date: ""));
        }
    }

    1 usage  Caljorb
    public static synchronized LeaderboardViewModel getLeaderboardViewModel() {
        if (leaderboardViewModel == null) {
            leaderboardViewModel = new LeaderboardViewModel();
        }
        return leaderboardViewModel;
    }
```

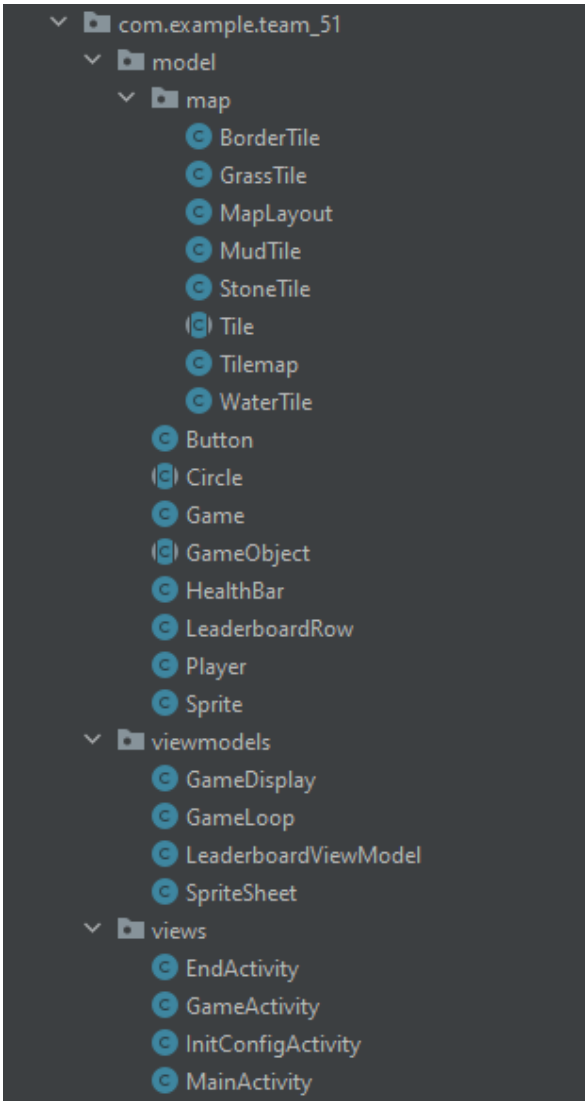
```
2 usages  Caljorb
public class EndActivity extends AppCompatActivity {
    22 usages
    private LeaderboardViewModel leaderboardViewModel;
    2 usages
    private ArrayList<LeaderboardRow> leaderboardRows;
    3 usages
    private long score;
    3 usages
    private String name;
    3 usages
    private String date;
    5 usages
    private boolean retried;
    Caljorb
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_end_screen);

        leaderboardViewModel = LeaderboardViewModel.getLeaderboardViewModel();
        leaderboardRows = getIntent().getParcelableArrayListExtra( name: "leaderboard");
        retried = getIntent().getBooleanExtra( name: "retried", retried);
        if (retried) { // only update viewmodel when have retried before
            leaderboardViewModel.setRows(leaderboardRows);
        }
```

Singleton Explanation

To properly implement the Singleton design for the leaderboard, we first created the LeaderboardRow class, in which a single LeaderboardRow consists of a Name, a Score, and a Date. These are used to create the LeaderboardViewModel. The LeaderboardViewModel is where singleton was implemented, as we do not want to store more than one Leaderboard at a time. To do this, we made the constructor private, and added the getLeaderboardViewModel method to allow for using an instance of this class. This method is synchronized to prevent threads from creating multiple instances, and it checks if it has been created yet with the if-statement, checking if leaderboardViewModel is null. This comes into play in the EndActivity class, where we instantiate the LeaderboardViewModel class to use to create the leaderboard on the end screen; however, we use singleton here because the user has the option to retry and start again from the beginning. In this case, we want to use the same leaderboard, so singleton lets us reuse the one we originally created. If we ever want to update the LeaderboardViewModel, we use the set methods defined in the class to update the LeaderboardRows that make it up, without ever making a new LeaderboardViewModel.

MVVM Architecture Evidence

1. Folder Structure



2. Model Example

```
public class LeaderboardRow implements Parcelable {
    6 usages
    private String name;
    6 usages
    private long score;
    6 usages
    private String date;

    3 usages  Caljorb
    public LeaderboardRow(String name, long score, String date) { // store 3 elements for a row
        this.name = name;
        this.score = score;
        this.date = date;
    }
```

```
 Caljorb
public String getName() { return name; }

7 usages  Caljorb
public long getScore() { return score; }

5 usages  Caljorb
public String getDate() { return date; }

 Caljorb
public void setName(String name) { this.name = name; }

no usages  Caljorb
public void setScore(long score) { this.score = score; }

no usages  Caljorb
public void setDate(String date) { this.date = date; }
```

3. ViewModel Example

```java
public class LeaderboardViewModel extends ViewModel {
    10 usages
    private ArrayList<LeaderboardRow> leaderboardRows;
    3 usages
    private static LeaderboardViewModel leaderboardViewModel;

    1 usage  Caljorb
    private LeaderboardViewModel() {
        leaderboardRows = new ArrayList<>( initialCapacity: 5);
        for (int i = 0; i < 5; i++) {
            leaderboardRows.add(new LeaderboardRow( name: "",  score: 0,  date: ""));
        }
    }

    3 usages  Caljorb
    public static synchronized LeaderboardViewModel getLeaderboardViewModel() {
        if (leaderboardViewModel == null) {
            leaderboardViewModel = new LeaderboardViewModel();
        }
        return leaderboardViewModel;
    }

    2 usages  Caljorb
    public void setRows(ArrayList<LeaderboardRow> leaderboardRows) {
        this.leaderboardRows = leaderboardRows;
    }

    1 usage  Caljorb
    public void addRow(LeaderboardRow leaderboardRow) {
        leaderboardRows.add(leaderboardRow);
        // adds to end, call sort after
    }

    1 usage  Caljorb
    public void sortRows() { // use for easy sorting
        Caljorb
        leaderboardRows.sort(new Comparator<LeaderboardRow>() {
            Caljorb
            @Override
            public int compare(LeaderboardRow leaderboardRow, LeaderboardRow t1) {
                int score = (int) (t1.getScore() - leaderboardRow.getScore());
                if (score == 0) {
```

4. View Example

```
public class GameActivity extends AppCompatActivity {
    2 usages
    private static Context context;
    2 usages
    private int character;
    2 usages
    private int hp;
    2 usages
    private String name;
    2 usages
    private final long startScore = 60000;
    1 usage
    private CountDownTimer countDownTimer;
    4 usages
    private Game game;
    Caljorb
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        GameActivity.context = getApplicationContext();

        // get values to make game
        hp = getIntent().getIntExtra( name: "hp",  defaultValue: 100);
        name = getIntent().getStringExtra( name: "name");
        character = getIntent().getIntExtra( name: "character",  defaultValue: 1);

        // make game
        game = new Game( context: this, hp, name, character, startScore);
        setContentView(game);
```

5. Class List
    a. Models
        i. BorderTile
        ii. GrassTile
        iii. MapLayout
        iv. MudTile
        v. StoneTile
        vi. Tile
        vii. Tilemap
        viii. WaterTile
        ix. Button
        x. Circle
        xi. Game
        xii. GameObject
        xiii. HealthBar
        xiv. LeaderboardRow
        xv. Player

xvi. Sprite

b. ViewModels
    i. GameDisplay
    ii. GameLoop
    iii. LeaderboardViewModel
    iv. SpriteSheet
c. Views
    i. EndActivity
    ii. GameActivity
    iii. InitConfigActivity
    iv. MainActivity
    v. activity_end_screen
    vi. activity_init_config
    vii. activity_start_screen

6. How we used MVVM:

   To use MVVM, we first started by organizing the files we had from sprint 1 into the model, viewmodel, and view folders. All of the activity classes that displayed screens were placed into the view folder but player was placed into model. From there, we started creating the tile classes and related fields. These were all placed in the model folder as they hold the data for the tiles that construct the maps. From there, we created SpriteSheet, GameLoop, and GameDisplay, all of which are viewmodels that handle the logic for a view. For this sprint, these viewmodels were used to help handle displaying the game screen. Another example of MVVM coming into play was for creating the leaderboard. The leaderboard is made of leaderboardRows, which store a name, score, and date. From there, we made a leaderboardViewModel that handles the logic of a leaderboard, including sorting the scores, adding scores, and removing scores. All of this was displayed on the View EndActivity.