UP930788 DnD Adventure Game APP

Design:

The decision map design has been included in a separate PDF folder as it is too large to be included here. The design map illustrates all the links to different nodes to showcase the story's flow; the map represents a multipath adventure game with three distinct storylines. The primary purpose of this was to create a byte size, scaled-down version of how a Dungeons and Dragons game could be designed.

Some node paths are labelled ‘SUCCESS’ or ‘FAILURE’ these nodes will act as if a ‘roll’ aka a dice roll has been cone to decide whether the action chosen will succeed or not. The chances of success are also aptly affected by the character statistics, e.g. strength or class of a character helping to shape the likelihood of success or failure is decided which inturn the next part of the story, e.g. failure can lead to death or denial of access from a story branch.

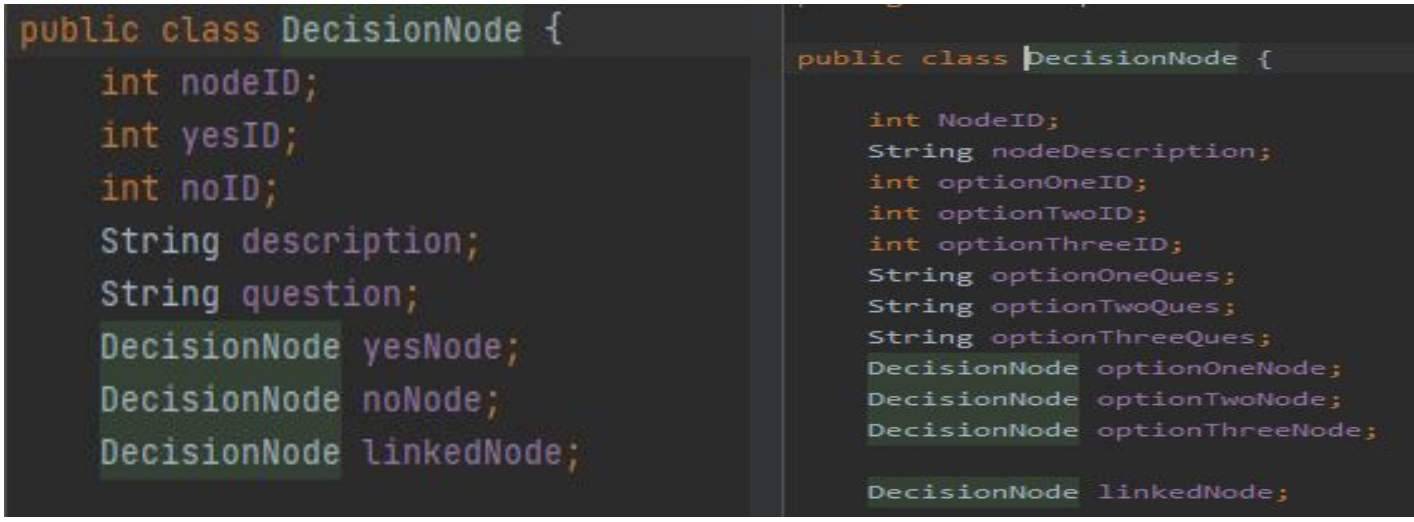
The application’s goal is to help potential players interested in dungeons and dragons to understand the game’s basics and play out different aspects of the game, e.g. combat, exploration, and detective work.

Changes to classes:

I made changes to the DecisionNode and DecisionMap classes to better fit my design from the java classes. Additionally, I did not use the Utils, and DecisionMapTest classes as they were designed for a console application would not make sense with my front end platform choice of Android Studio.

DecisionNode changes:

The image on the left is from the given DecisonNode class. In contrast, the image on the right is my modified version of it. The main difference is the number of attributes a node can have has been increased, and variable names have been changed to reflect my application design better as it does not contain yes or no choices. Instead, they are labelled as 'options', each node can have a maximum of 3 options the user can choose from as any given point in the story. Other than these changes, the rest of the class generally the same as the given as the class only has getter and setter methods.



```
public class DecisionNode {
    int nodeID;
    int yesID;
    int noID;
    String description;
    String question;
    DecisionNode yesNode;
    DecisionNode noNode;
    DecisionNode linkedNode;
```

```
public class DecisionNode {

    int NodeID;
    String nodeDescription;
    int optionOneID;
    int optionTwoID;
    int optionThreeID;
    String optionOneQues;
    String optionTwoQues;
    String optionThreeQues;
    DecisionNode optionOneNode;
    DecisionNode optionTwoNode;
    DecisionNode optionThreeNode;

    DecisionNode linkedNode;
```

Decision Map changes :

The functions of both DecisionMap classes are the same but are done differently; the first being that rather than use the inbuilt Scanner class to get data from the CSV file. Therefore, I used a buffered reader; my reasons for this were related to the android studio being unable to find the CSV file's pathing, (I spent hours debugging trying to understand why). They effectively do the same job expect parsing of the data is possible directly through a Scanner. The two main methods affected were the DecisionMap and buildUnorderList, as seen below.

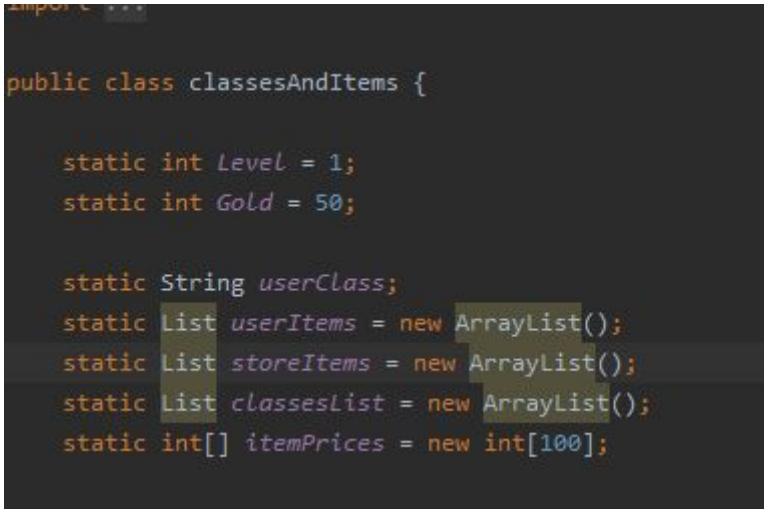
```
public DecisionMap(BufferedReader csvFile) {
    try {
        buildUnorderedList(csvFile);
    } catch (IOException e) {
        e.printStackTrace();
    }
    buildOrderedMap();

}

public void buildUnorderedList(BufferedReader csvFile) throws IOException {

    String line;
    DecisionNode node;
    int numOfNodes = 0;
    do {
        line = csvFile.readLine();
        try {
            node = buildNode(line);
            append(node);
        } catch (emptyLineErrorHandling emptyLineErrorHandling) {
            Log.w( tag: "CSVLineIsBlank",emptyLineErrorHandling);
        }
        numOfNodes++;
    }while(numOfNodes != 54);
}
```

Other methods that changed in this class were the buildNode and buildOrderMap as due to the DecisionNode class the being different they had to be adjusted accordingly to match the CSV file format. Therefore, it accounts for all the new variables and the regex expression of splitting each CSV line by commas edited so that commas inside quotation marks are not effected as some nodes contain information to drive the story, requiring punctuation including commas.

```
private DecisionNode buildNode(String line) throws emptyLineErrorHandling {
    if(line.equals("")){
        throw new emptyLineErrorHandling( input: "Oh no you have an empty line in your CSV!
    }
    String[] stringArray = line.split( regex: ",(?=(?:[^\"]*\"[^\"]*\")*[^\"]*$)");

    DecisionNode n = new DecisionNode();
```

Custom backend java classes:

Other than changes made to the given classes, I also made my backend java classes as they were needed to facilitate all the values required for the application to function and progress through the story. These include: gameProgression, classesAndItems and characterAtrributes. These classes only contain code java code with no front end android studio code. The classesAndItems and characterAttrbtues are comprised of just getter and setter like methods to hold values related to a player, such as the different classes they can choose and set, the amount of gold they may have, the items as well as attributes about the player. These values are all used to either calculate the storyline's path or are displayed to the player are different points in the application.



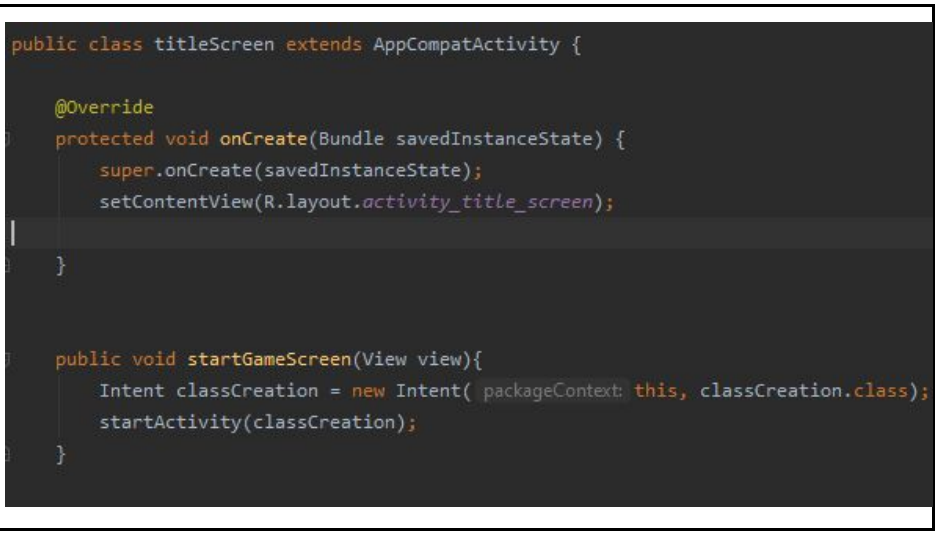
```
public class classesAndItems {

    static int Level = 1;
    static int Gold = 50;

    static String userClass;
    static List userItems = new ArrayList();
    static List storeItems = new ArrayList();
    static List classesList = new ArrayList();
    static int[] itemPrices = new int[100];
```

The class gameProgression is used to verify at what point in the story the player is currently at and if the decision they take requires any special calculation to decide if it is a success or failure and if a node needs to be edited to output a correct value/text depending on what class the player has picked then it is checked and done through this class.

Custom Frontend java Classes:

The other classes I have are all related to the front code, e.g. Textviews, ImageView, button clicks, toast messages. These classes are titleScreen, gameScreen, characterSheet, classCreation, attributeSelect and adventurersStore. The titleScreen is the first home screen the user sees when they start the application. It gives a brief introduction. When the start game button is pressed, the next activity displayed allows the user to choose one of three classes they want to be; the classCreation class handles this. Once a class has been picked, the user is taken to an activity allowing them to 'roll' 4 dices pick the best three results to add them together and assign that number to their character attributes. Once all attributes have been initialised and populated the player is shown their character sheet, something akin to a real-life DnD character sheet but scaled-down and the characterSheet class handles this. Only then can the player actually start to play through the different storylines displayed through the gameScreen class and then from within the gameScreen class can the players

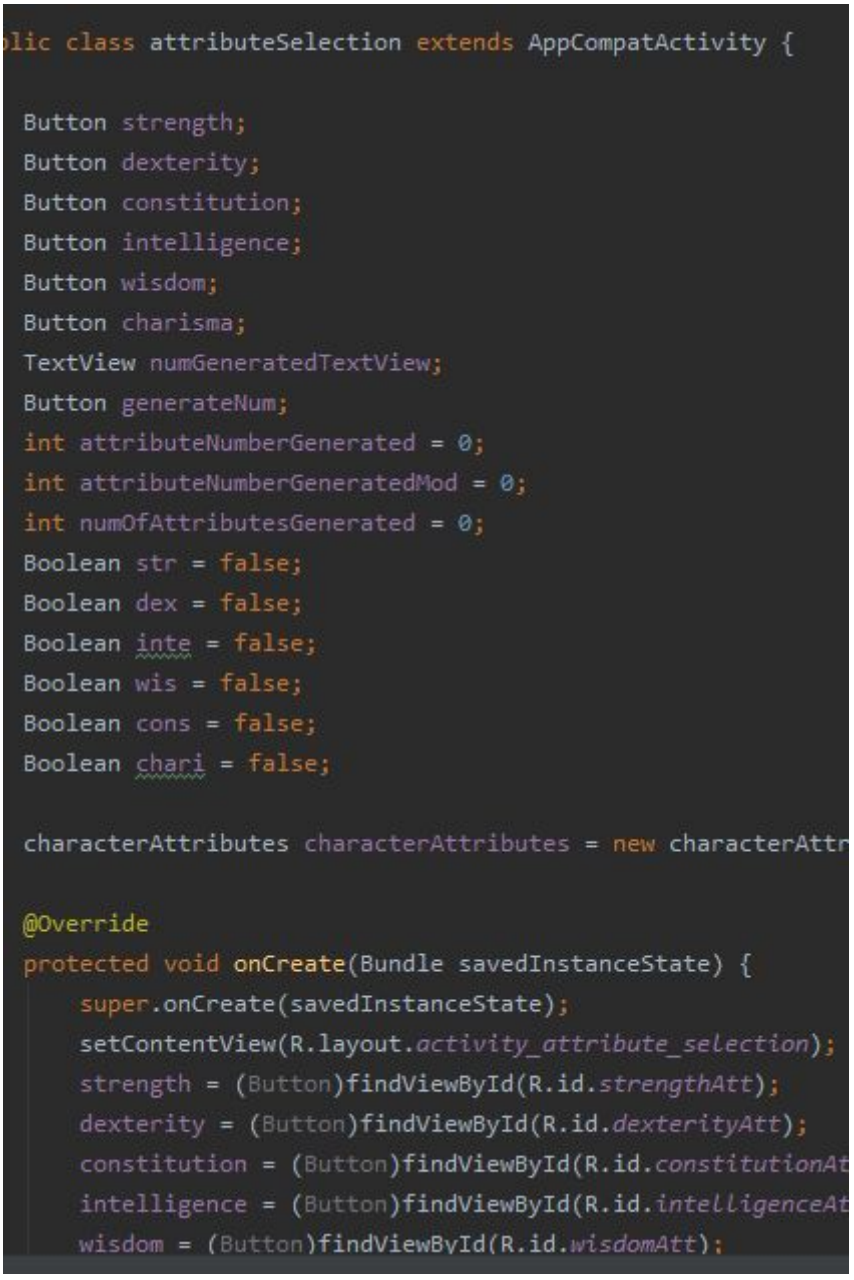
also access the adventurers' store which is handled by the player adventurersClass. When creating the front-end java classes, the idea was to initialize the required objects such as buttons and textviews and have the node processing and all other processing done by other non-activity classes not to become overcomplicated when changes need to be made visually.

titleScreen	

```java
public class titleScreen extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_title_screen);

    }


    public void startGameScreen(View view){
        Intent classCreation = new Intent( packageContext: this, classCreation.class);
        startActivity(classCreation);
    }
```

classCreation	

```
public class classCreation extends AppCompatActivity {

    Button Warrior;
    Button Mage;
    Button Rouge;

    @Override
    protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_class_creation);
        Warrior = (Button)findViewById(R.id.warriorBtn);
        Mage = (Button)findViewById(R.id.mageBtn);
        Rouge = (Button)findViewById(R.id.rougeBtn);

        Warrior.setText("Warrior");
        Mage.setText("Mage");
        Rouge.setText("Rouge");

        classesAndItems.makeClasses();
        classesAndItems.makePrices();

        Warrior.setOnClickListener(new View.OnClickListener() {
            @Override

            public void onClick(View v) { warriorSelected(); }
        });

        Mage.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { mageSelected();}
```

attributeSelection	

```
blic class attributeSelection extends AppCompatActivity {

    Button strength;
    Button dexterity;
    Button constitution;
    Button intelligence;
    Button wisdom;
    Button charisma;
    TextView numGeneratedTextView;
    Button generateNum;
    int attributeNumberGenerated = 0;
    int attributeNumberGeneratedMod = 0;
    int numOfAttributesGenerated = 0;
    Boolean str = false;
    Boolean dex = false;
    Boolean inte = false;
    Boolean wis = false;
    Boolean cons = false;
    Boolean chari = false;

    characterAttributes characterAttributes = new characterAttr

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_attribute_selection);
        strength = (Button)findViewById(R.id.strengthAtt);
        dexterity = (Button)findViewById(R.id.dexterityAtt);
        constitution = (Button)findViewById(R.id.constitutionAt
        intelligence = (Button)findViewById(R.id.intelligenceAt
        wisdom = (Button)findViewById(R.id.wisdomAtt);
```

characterSheet

```java
public class characterSheet extends AppCompatActivity {

    Button continueBtn;
    TextView strenTextView;
    TextView intellTextView;
    TextView wisdomTextView;
    TextView dexTextView;
    TextView constTextView;
    TextView charisTextView;
    TextView goldTextView;
    TextView levelTextView;
    TextView classTextView;
    TextView itemsTextView;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_character_sheet);

        strenTextView = (TextView)findViewById(R.id.strenText);
        intellTextView = (TextView)findViewById(R.id.intellText);
        wisdomTextView = (TextView)findViewById(R.id.wisdomText);
        dexTextView = (TextView)findViewById(R.id.dexterText);
        constTextView = (TextView)findViewById(R.id.constText);
        charisTextView = (TextView)findViewById(R.id.charisText);
        continueBtn = (Button)findViewById(R.id.continueBtn);
        goldTextView = (TextView)findViewById(R.id.goldText);
        levelTextView = (TextView)findViewById(R.id.levelText);
        classTextView = (TextView)findViewById(R.id.classTextView);
        itemsTextView = (TextView)findViewById(R.id.itemsText);
```

gameScreen	

```
public class gameScreen extends AppCompatActivity {

    Button optionOne;
    Button optionTwo;
    Button optionThree;
    TextView nodeDesc;
    ImageView image;
    static DecisionNode previousNode = null;

    int[] decisionRollIDs = {1000,1001,2000,2001,2002,3000,3001,7};
    int[] textRollIDs = {27,11,12};
    String newNodeDesc;
    gameProgression gameProgression = new gameProgression();


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_game_screen);

        optionOne = (Button)findViewById(R.id.optionOne);
        optionTwo = (Button)findViewById(R.id.optionTwo);
        optionThree = (Button)findViewById(R.id.optionThree);
        nodeDesc = (TextView)findViewById(R.id.nodeDescription);
        nodeDesc.setMovementMethod(new ScrollingMovementMethod());
        image = (ImageView)findViewById(R.id.gameImage);


        DecisionMap perec = new DecisionMap(getCSV());
        navigate(perec);
    }
```

adventurersStore	

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_adventurers_store);



    classesAndItems.setStoreItems();
    classesAndItems.makePrices();
    List storeItems = classesAndItems.getStoreItems();
    int[] itemPrices = classesAndItems.getItemPrices();

    GoldTextView = (TextView)findViewById(R.id.goldText);
    GoldTextView.setText(String.valueOf(classesAndItems.getGold()));

    titleTextView = (TextView)findViewById(R.id.titleTextView);

    itemOne = (TextView)findViewById(R.id.itemOneTextView);
    itemTwo =  (TextView)findViewById(R.id.itemTwoTextView);
    itemThree = (TextView)findViewById(R.id.itemThreeTextView);
    itemFour = (TextView)findViewById(R.id.itemFourTextView);

    itemOnePriceView = (TextView)findViewById(R.id.itemOnePrice);
    itemTwoPriceView = (TextView)findViewById(R.id.itemTwoPrice);
    itemThreePriceView = (TextView)findViewById(R.id.itemThreePrice);
    itemFourPriceView = (TextView)findViewById(R.id.itemFourPrice);



    titleTextView.setMovementMethod(new ScrollingMovementMethod());

    buyItemOne = (Button)findViewById(R.id.buyItemOneBtn);
    buyItemTwo = (Button)findViewById(R.id.buyItemTwoBtn);
    buyItemThree = (Button)findViewById(R.id.buyItemThreeBtn);
```

Custom exception handlers:

I created the custom handler class emptyLineErrorHandling to check whether the buffer reader read an empty line from the CSV file as when creating a CSV file. The exception handler was created as there was much editing to the file format and correcting the file. Sometimes, empty lines either crashed the program or incorrectly linked nodes together, which is why this handler exists. The error handler is called when a new node is being built made. Other than this, I also used inbuilt java exception handlers such as IOExceptions for the method buildUnorderedNode as this takes the buffered reader as an input, which can sometimes have other issues.

```java
public void buildUnorderedList(BufferedReader csvFile) throws IOException {

    String line;
    DecisionNode node;
    int numOfNodes = 0;
    do {
        line = csvFile.readLine();
        try {
            node = buildNode(line);
            append(node);
        } catch (emptyLineErrorHandling emptyLineErrorHandling) {
            Log.w( tag: "CSVLineIsBlank",emptyLineErrorHandling);
        }
        numOfNodes++;
    }while(numOfNodes != 54);
}
```

```
W/CSVLineIsBlank: com.example.introtodnddecisiongame.emptyLineErrorHandling: Oh no you have an empty line in your CSV!
    at com.example.introtodnddecisiongame.DecisionMap.buildNode(DecisionMap.java:118)
    at com.example.introtodnddecisiongame.DecisionMap.buildUnorderedList(DecisionMap.java:62)
```

```java
private DecisionNode buildNode(String line) throws emptyLineErrorHandling {
    if(line.equals("")){
        throw new emptyLineErrorHandling( input: "Oh no you have an empty line in your CSV!");
    }
```

```java
package com.example.introtodnddecisiongame;


public class emptyLineErrorHandling extends Exception {
    public emptyLineErrorHandling(String input) { super(input); }
}
```

Video Link: https://youtu.be/jTZhG7LE7cI