S3

Mom and Dad's Pizza Pad Pizza Adventure

Detailed Design and Test Plan

Team C - COMP 3663 X1
Scott Barnett 100123218, Jimmy Flemming 100116643,
Liam MacKinnon 100114393, Martin Main 100110854
March 8, 2016

Table of Contents

Executive Overview/Summary

The purpose of this document is to give a detailed description of the design of Mom and Dad's Pizza Pad's Pizza Adventure application. It provides a full specification of all packages, modules, classes, class attributes, and class methods. The decomposition and interaction between these modules is given as well. Testing procedures for the individual classes as well as for the system as a whole are included. For the purpose of organization, the document is split into two sections, one being the detailed design document, and the other being the testing procedures document.

Section 1 - System Design Document

The System Design Document is provided to comprehensively describe all architectural details and all classes, methods, and functions used in the application. Descriptions are provided when necessary about class attributes and methods. Some Pseudo code and interaction diagrams are provided to understand the specific logical structures used in some of the method descriptions. A class hierarchy diagram is provided to understand the overarching architectural structure of the application. An implementation schedule is provided as a part of this document.

Section 2 - System Test Plan

The System Test Plan is provided with the purpose to provide an comprehensive overview of the testing methodology used in the system, including a number of sample unit tests, integration tests, and system tests. The testing processes and approach to the various stages of testing will be discussed in sufficient detail. Scheduling and resource allocation for the testing procedures will be provided as well.

If there are any questions or concerns about anything provided in the document, please do not hesitate to contact our customer communication manager Liam MacKinnon at 114393m@acadiau.ca

-La Corporación C

System Design Document

1. Introduction

1.1 Purpose

The purpose of the System Design Document is to comprehensively describe all architectural details and all classes, methods, and functions used in the application. The complete document should be sufficient documentation to hand to a programming team to fully implement and test the system.

1.2 Scope

The scope of this document will include descriptions and details of all classes, methods, attributes, and architecture decomposition. Additional descriptions are added when necessary about class attributes and methods. Some pseudocode and interaction diagrams are provided to understand the specific logical structures used in some of the method descriptions. A class hierarchy diagram is provided to give the readers an understanding of the overarching architectural structure of the system. Comprehensive implementation and test schedules are included to give a clear description of when each implementation deliverable will be completed.

1.3 Definitions, Acronyms, and Abbreviations

GUI

Acronym for “Graphical user interface” and refers to the type of user interaction system that appears as various buttons and graphics on a screen which users can interact with using their mouse or keyboard.

GUMP

Acronym for “Giant Ultra Mutant Pepperoni” and is a part of the game
Mom and Dad's Pizza Pad's Pizza Adventure.

JRE

Acronym for “Java Runtime Environment”, which is a program that needs to be installed on the computers running the system. This program is already installed on most modern computers. It allows programs written in the programming language “Java” to be run on the computer. This runtime environment has many versions, and a sufficiently late version of the runtime environment will need to be installed on the computers running the game.

LORP
Acronym for “Lump of radioactive pepperoni” and is a part of the game Mom and Dad's Pizza Pad's Pizza Adventure.

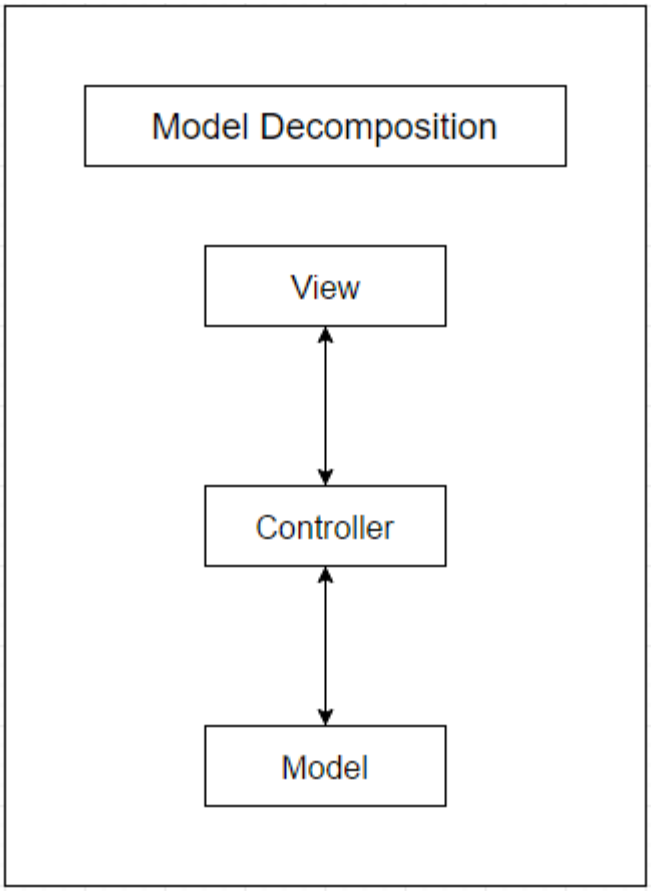
OS
Acronym for “Operating System”, and refers to the base system installed on any computer from which all other programs are run.

UI
Acronym for “User interface” and refers to any system that a user of the system would use
to interact with the system.

3. Decomposition Description

3.1 Module Decomposition

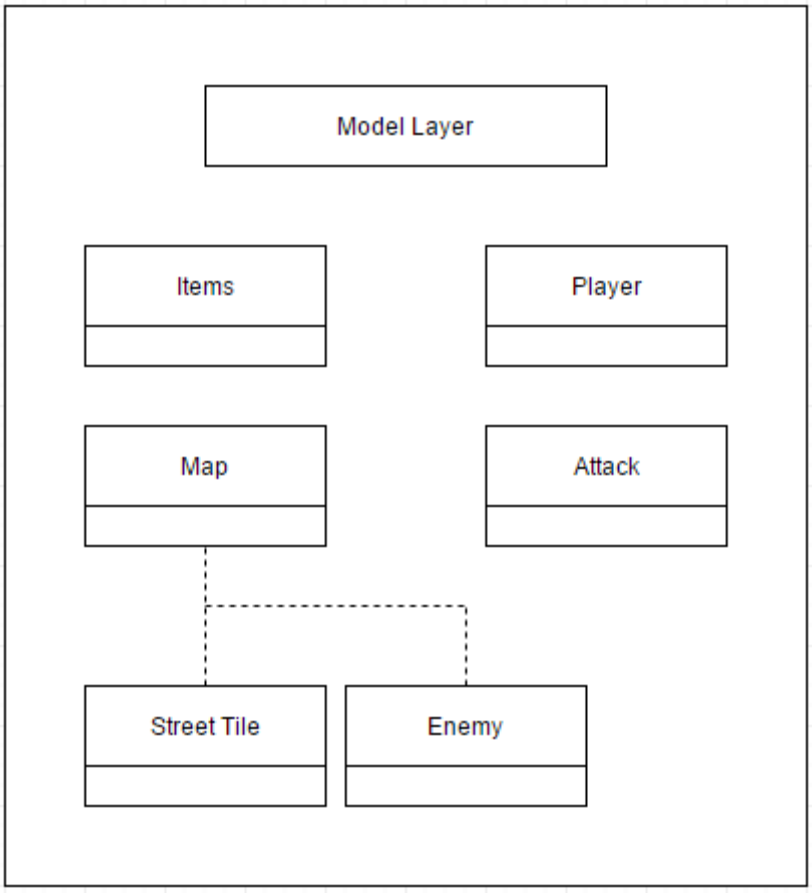
Mom And Dad's Pizza Pad Pizza Adventure will have three layers: the Model layer, the View layer, and the Control layer. The model is manipulated by the controller layer, and uses the controller input to update the view. The view is the layer that holds the GUI components, and is sent updates from the model layer. The controller layer takes input from peripherals, mostly a keyboard or mouse in our case, and sends that information to the model layer.



3.1.1 Model Layer Decomposition

The model is manipulated by the controller layer, and uses the controller input to update the view. The controller layer will contain classes for:

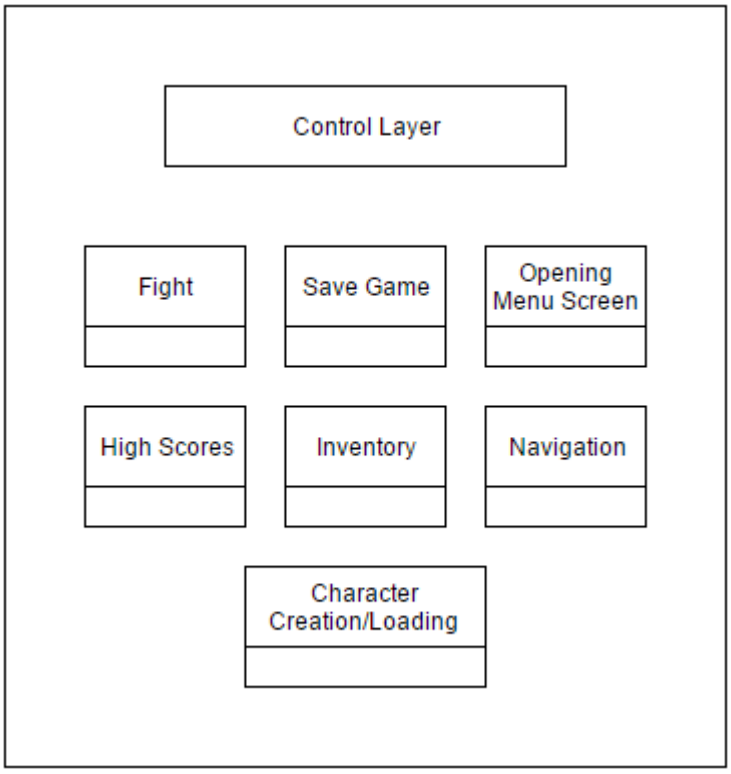
1. Map
2. Player
3. Game Stats
4. Items



3.1.2 Control Layer Decomposition

The control layer takes input from peripherals, mostly a keyboard or mouse in our case, and sends that information to the model layer. The controller layer will contain classes for:

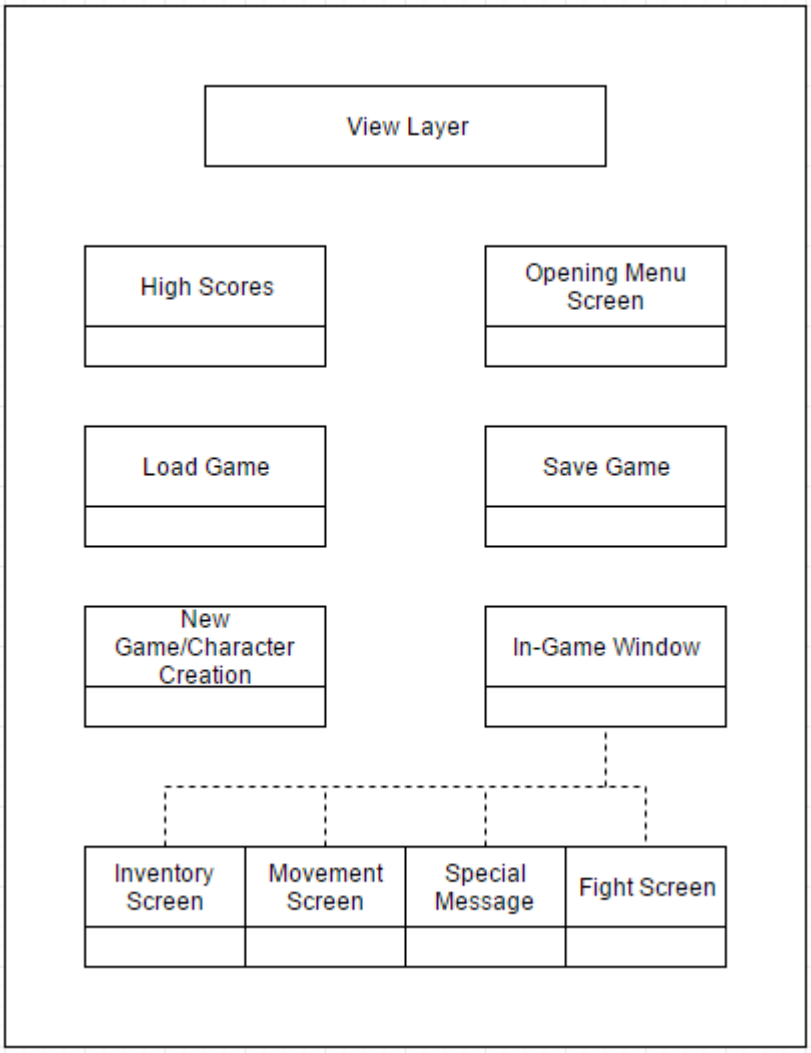
1) Fight
2) Character Creation/Loading
3) Save Game
4) Opening Menu Screen
5) High Scores
6) Inventory
7) Navigation



3.1.3 View Layer Decomposition

The view is the layer that holds the GUI components, and is sent updates from the model layer. The view layer will contain classes for the:

1) Opening Menu Screen
2) Load Game/Character Creation Screen
3) In-Game Window



3.2 Overall design approach

We are using the Model, View, Controller architecture to design our system, in which major components will fall under one of the three sections of Model, View and Controller.

All data level components of the system will be contained under the Model class. The model class will need to contain a consistent representation of the state of the game at any given time. It will contain all the “objects” of the game, such as Street Tile, Player, and Items. Each of these classes will contain many attributes, and the methods will be linked specifically to that class and the access and manipulation of the data in that class.

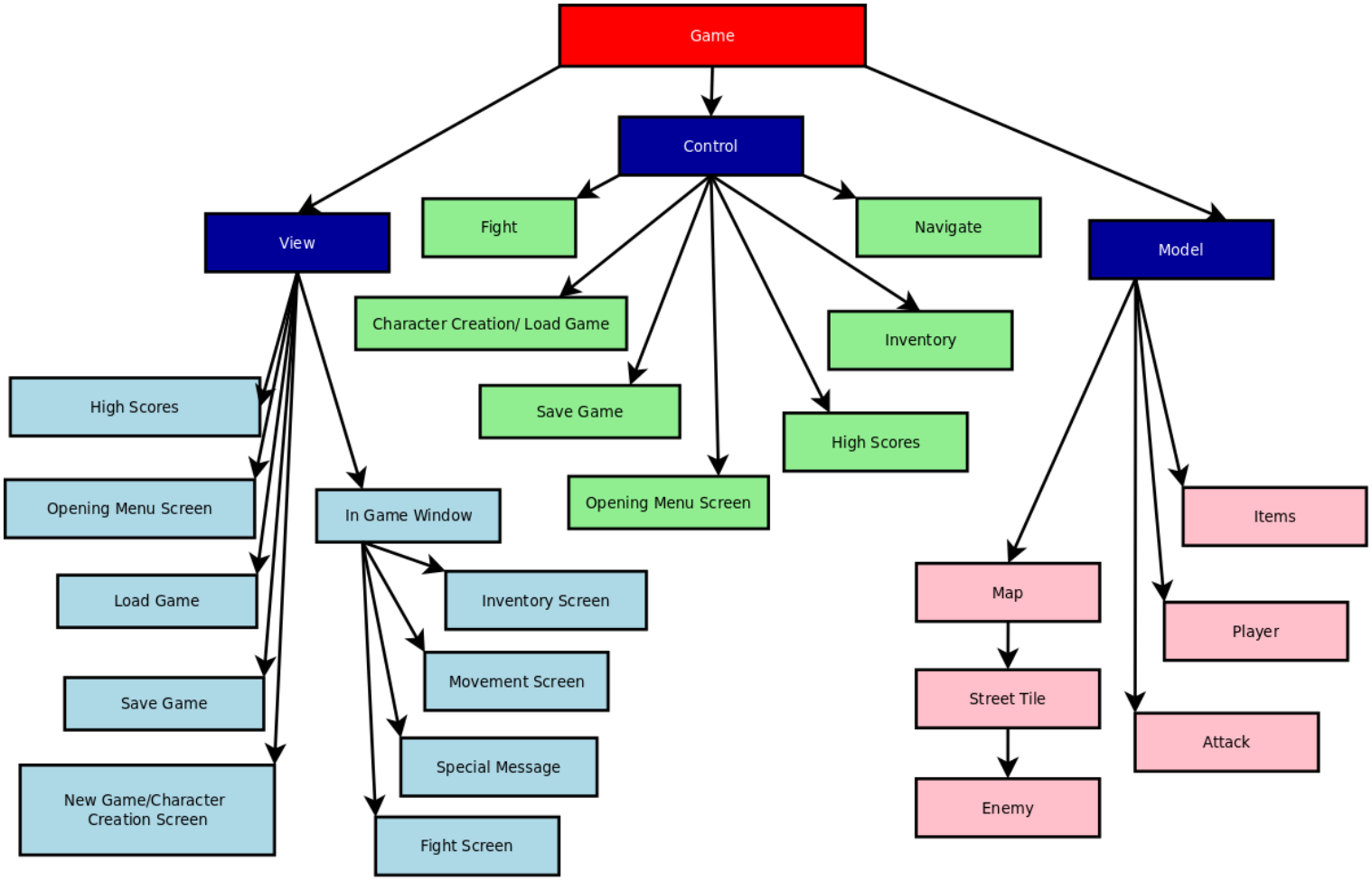
All visual aspects of the system will be contained under the view class. This will include the construction of GUI’s with all text fields, buttons, and other graphical components. The classes contained in the view package will not contain logic, but will simply be the outer skin of the application. This will be updated by the controller and the model to represent a consistent state of the system to the user at all times.

All control aspects of the system will be contained under the Controller package. These will include all event handlers for keyboard actions at any state in the game, as well as the click handlers for buttons and other fields in the game. The classes in this package will have fewer attributes, and more methods to control the data flow in the game and call the methods in the classes in the model package to satisfy changes made to the model.

For example of how the pieces fit together, consider the action of using Mr. Michel’s Map from the user’s inventory:

1. The inventory is displayed by a class in the view
2. The user clicks on the icon for the map, and a controller class sends a message to the view, to highlight the area clicked on
3. The user clicks “OK”, and the controller sends messages to the view to hide the inventory menu, and sends a message to the items class in Model, to use Mr. Michel’s map. It also sends a message to decrement the number of maps being held in the model.
4. The items class in the model sends a message to the map class to reveal all areas near to the player, thus using the item.
5. The view is updated by the map class to show the revealed areas.

3.2.1 Class Hierarchy Diagram



3.3 Decomposition into major components

3.3.1 View Package

3.3.1.1 Classes

The View package will contain the following classes:

- High Scores
- Opening Menu Screen
- Load Game
- Save Game
- New Game/Character Creation Screen
- In Game Window
- Inventory Screen
- Movement Screen
- Special Message
- Fight Screen

3.3.1.2 Instances

- There will be one instance of In Game Window.
- In Game Window may (or may not) contain an instance of Inventory Screen
- In Game Window may (or may not) contain an instance of Movement Screen
- In Game Window may (or may not) contain an instance of Special Message
- In Game Window may (or may not) contain an instance of Fight Screen

3.3.2 Control Package

3.3.2.1 Classes

The Control package will contain the following classes:

- Fight
- Character Creation/Load Game
- Save Game
- Opening Menu Screen
- High Scores
- Inventory
- Navigate

3.3.2.2 Instances

- There will only be one instance of Fight at a time
- There will only be one instance of Creation Character/Load Game at a time
- There will only be one instance of Save Game at a time
- There will only be one instance of Opening Menu Screen at a time
- There will only be one instance of High Scores at a time
- There will only be one instance of Inventory at a time
- There will only be one instance of Navigate at a time

3.3.3 Model Package

3.3.3.1 Classes

The Model package will have the following classes:

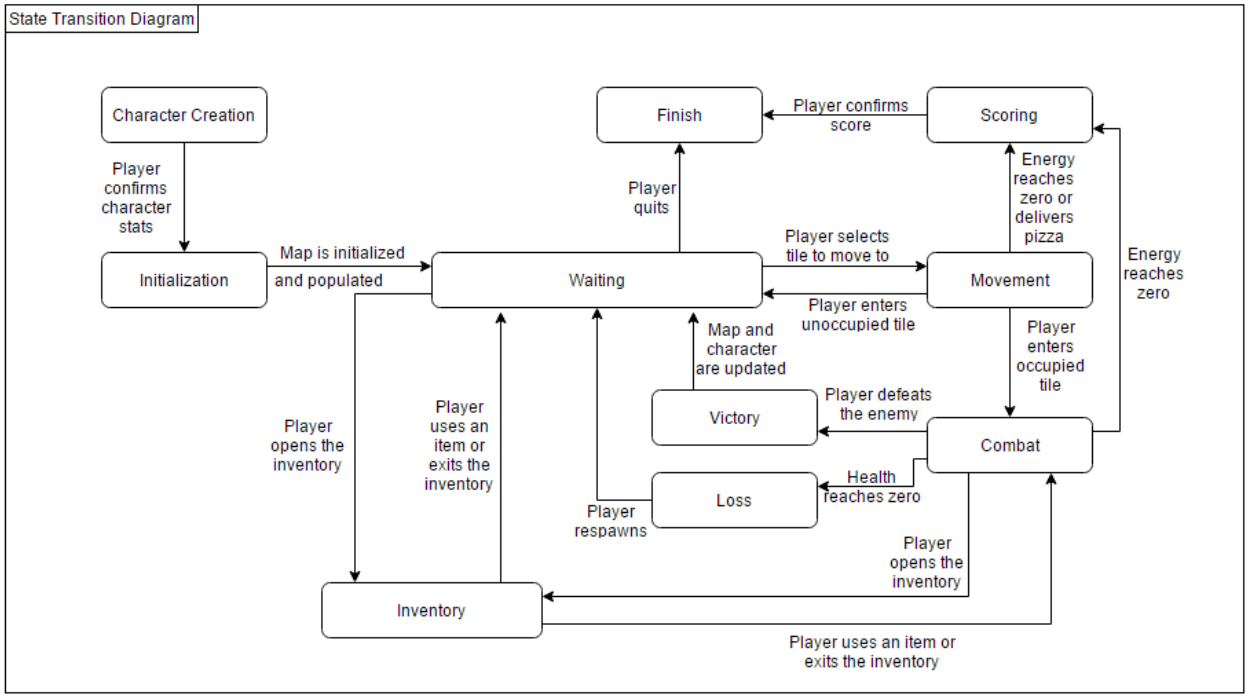
- Map
- Street Tile
- Enemy
- Items
- Player
- Attack

3.3.3.2 Instances

- There will only be one instance of Map.
- There will only be one instance of Player.
- Map will contain several instances of Street Tile.
- Street Tile may (or may not) contain an instance of Enemy.
- Street Tile may (or may not) contain an instance of Player.
- Street Tile may (or may not) contain an instance of Items.
- Player may (or may not) contain several instances of Items.

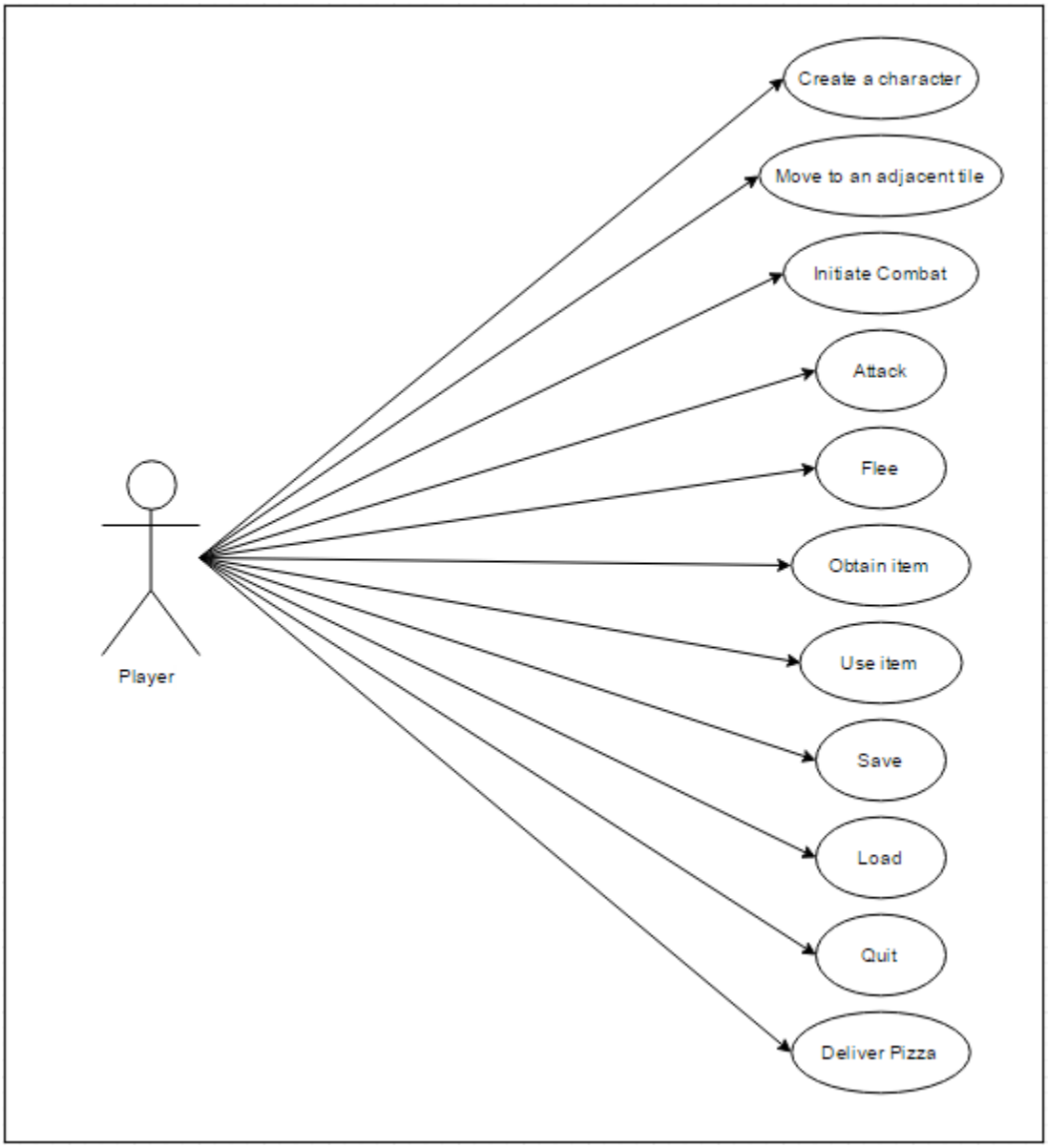
3.4 State Model Decomposition

Mom and Dad's Pizza Pad Pizza Adventure has the following states:
1) *Character Creation* - User is creating the player
2) *Initialization* - Map is being set up and populated
3) *Waiting* - Awaiting user input
4) *Movement* - The player is moving
5) *Combat* - The player is in combat
6) *Victory* - Player defeats the enemy
7) *Loss* - Player's health reaches zero
8) *Inventory* - The player is viewing the inventory
9) *Scoring* - The player delivers the pizza or runs out of energy
10) *Finish* - Player completes the game or quits



3.5 Use Case Decomposition

Mom and Dad's Pizza Pad Pizza Adventure uses the following use cases:



4. Dependencies Description

4.1 Interlayer Dependencies

All three main layers of the MVC architecture are packaged within the game layer. The control layer is the only layer that is not generally influenced by the other layers, since it primarily accepts user input from peripherals. Both other layers are dependent on the control layer. The view layer changes what is seen on the screen when its methods are called by the control layer, or the model (which the view is also dependent on). The model layer is dependent on the control layer to make changes to the attributes of entities, such as the main character, or the inventory.

When the model makes changes to its entities, they are sometimes updated in the view. For example, if a user wants to use an item in their inventory, the model is called to reflect a decrement to the item quantity, and to alter the player's attributes based on the item type. Once that item is used and all attributes are updated, the view must be updated to reflect a visible change on the screen, in the quantity of the item, and on the stats screen for the player.

5. Interface Description

5.1 Interface for Model Package

The Model package is influenced by the Control package. It can influence the View Package. This package holds the data for the game entities and performs necessary calculations when called upon.

5.2 Interface for Control Package

The Control package influences the Model package and View package. It can't be influenced by any other package. This package controls the game, changing the view, initializing events and calling on the models to perform actions.

5.3 Interface for View Package

The View package is influenced by both the Control package and the Model package. It can't influence any other package. This package holds all the gui components. it is accessed by the control and model package to update the game screen in certain situations.

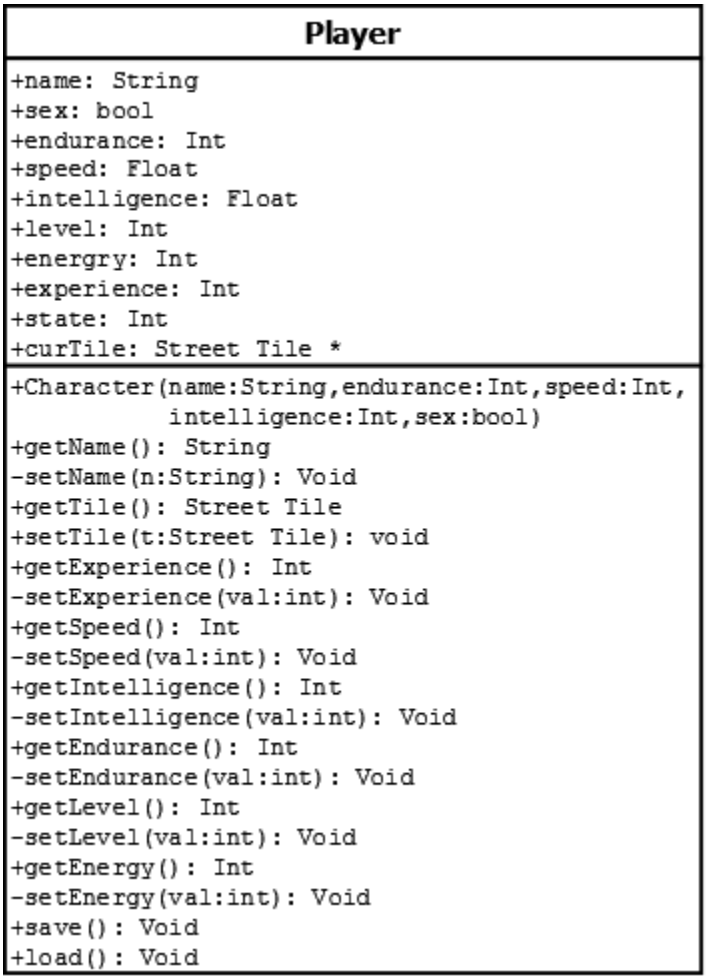
6. Detailed Design

6.0 Introduction

The purpose of the detailed design documentation is to completely describe all of the implementation details of the packages, classes, methods, and functions, etc used in the application. The purpose of this document is to be a comprehensive blueprint of the software system, which a programming team could use to completely develop a working system. The packages and classes are described in detail, providing attributes, methods, and any other relevant information, to give a complete understanding of what every component of the system should do.

6.1 Model Package

6.1.1 Player Class



6.1.1.1 Attributes

`String name`

The player's name.

`boolean sex`

The player's gender

`StreetTile * curTile`

The player's current location

`int endurance`

Number of points in the endurance stat

`int speed`

Number of points in the speed stat

`int intelligence`

Number of points in the intelligence stat

`int strength`

Number of points in the strength stat

`int level`

Player's current level

`int energy`

Player's current energy

`int experience`

Player's experience

6.1.1.2 Methods

`public character(string name, int endurance, int speed, int intelligence, int strength)`

Creates a player character with the given values

`public String getName()`

Get the player's name

`private void setName(string n)`

Set the player's name

`public StreetTile getTile()`

Get the player's current location

`private void setTile(StreetTile t)`

Set the player's current location

`public int getExperience()`

Get the player's experience

`private void setExperience(int val)`

Set the player's experience

`public int getStrength()`

Get the player's strength

`private void setStrength(int val)`

Set the player's strength

```
public int getSpeed()
```

Get the player's Speed

```
private void setSpeed (int val)
```

Set the player's Speed

```
public int getIntelligence()
```

Get the player's Intelligence

```
private void setIntelligence()int val
```

Set the player's Intelligence

```
public int getEndurance()
```

Get the player's Endurance

```
private void setEndurance(int val)
```

Set the player's Endurance

```
public int getLevel()
```

Get the player's Level

```
private void setLevel(int val)
```

Set the player's Level

```
public int getEnergy()
```

Get the player's Energy

```
private void setEnergy(int val)
```

Set the player's Energy

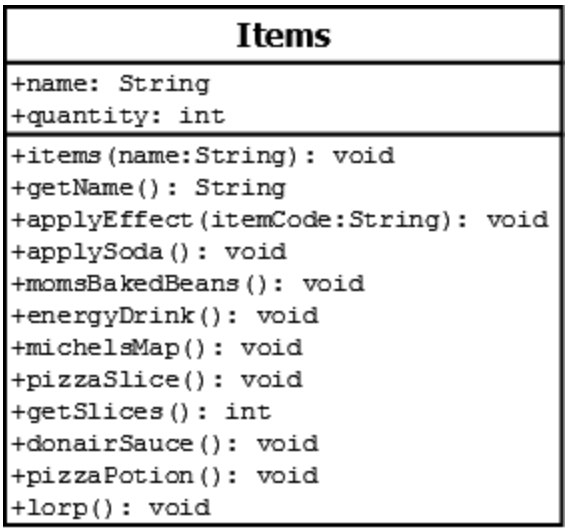
6.1.1.3 Instance Details

There will only be one instance of this class at a time.

6.1.1.4 Pseudo Code for *leveling up*

```
If experience is above next level threshold
     Set level to level plus one
     Set experience to zero
Increase player stats
```

6.1.2 Items Class



6.1.2.1 Attributes

`String name`

The name of the item

`int quantity`

How many of each item the player has

6.1.2.2 Methods

`public void item(String name)`

Constructor

`public String getName()`

Return the name of the item

`public void applyEffect(String itemCode)`

Case statement to decide what method to handle each item

`public void applySoda()`

Get the players energy level, replenish it partially, then set it

`public void momsBakedBeans()`

eSpeed = enemy.getSpeed()
decrease the enemy's speed so that the player has a greater chance for a successful attack

`public void energyDrink()`

get the players speed, increase it, and set it

`public void michelsMap()`

`public void pizzaSlice()`

increase the players current energy, and decrement the slices

`public int getSlices()`

return the number of pizza slices the player has compiled

`public void donairSauce()`

refill the energy bar completely

`public void pizzaPotion()`

change player state to angry Larry

`public void lorp()`

change player state

6.1.2.3 Instance Details

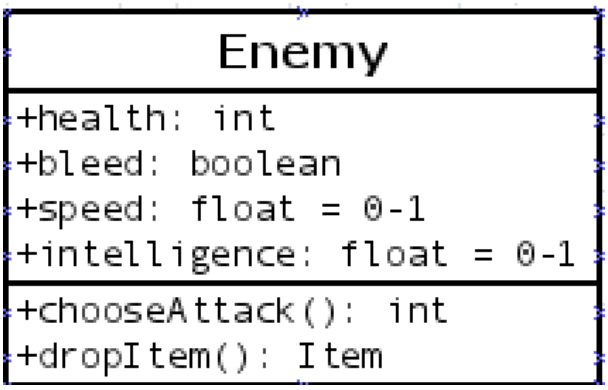
This class will be instantiated for as many items in the player's inventory at any time.

6.1.2.4 Pseudo Code for using an item

This class gets a call from the inventory view class

```
applyEffect(Item)
      switch (item)
            case: soda
                  applySoda()
            case: energyDrink
                  energyDrink()
            etc.
applySoda()
      Energy = Player.getEnergy()
      Player.setEnergy(Energy + x)
      //each item method effects the player stats in a different
way,
      //mostly using getters and setters
```

6.1.3 Enemy Class



6.1.3.1 Attributes

```
int health
```

total health points of the enemy

```
boolean bleed
```

Whether or not the enemy is losing health per turn

```
float speed //float between 0 and 1
```

Treated as a % chance to dodge an attack

```
float intelligence //float between 0 and 1
```

A % chance to have the enemy choose a weak attack against the player

6.1.3.2 Methods

```
public int chooseAttack()
```

random between 0 and 2, with intelligence altering the probability

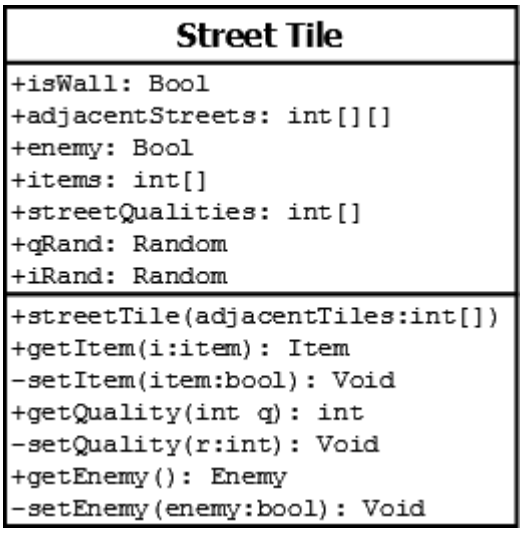
```
public Item dropItem()
```

random chance to drop a number of item possibilities

6.1.3.3 Instance Details

Maximum of one instance at a time. No instance of enemy while the player is in the map screen

6.1.4 Street Tile



6.14.1 Attributes

```
boolean isWall
```

True if the tile is a wall, false if the tile is a street

`boolean visible`

True if the player has been close to this tile before, false otherwise.

`int[][] adjacentStreets`

Direction (up, down, left, right) and street tile that can be traveled to from this tile.

`boolean enemy`

True if there is an enemy on the tile false if there isn't.

`int[] items`

Array of possible items for this tile (includes a blank entry for no item)

`int[] streetQualities`

Array of possible qualities the street can have (includes blank entry for no quality)

6.1.4.2 Methods

`public StreetTile(int[] adjacentTiles)`

Creates the street tile, takes an array of adjacent tiles to determine tile position

`public item getItem(item i)`

Retrieves item for the tile based on the argument passed to it

`private void setItem(bool i)`

Marks that an item has been picked up

`public int getQuality(int q)`

Gets the quality of the tile

`private void setQuality(int r)`

Sets quality of the tile based on the integer passed to it, uses the quality array to determine it

`public enemy getEnemy()`

Gets the enemy for the tile

`private void setEnemy(bool e)`

Sets an enemy for the tile that the player will have to fight.

`public randomQuality()`

Randomizer to decide on quality of the tile. This will be tuned to make a reasonable chance of each item being generated.

6.1.4.3 Instance Details

Only the current tile and adjacent tile instances will exist at a time new ones will be generated when a player moves tiles

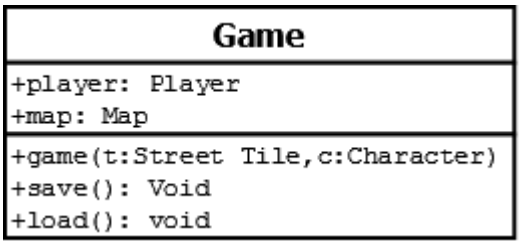
6.1.4.4 Pseudo Code *for getting an item*

```
If enemy != true
```

```
Get item corresponding to the item in the item array that is in the position determined by iRand. (Chance it could be nothing)
```

```
Gives the randomly selected item to the player.
```

6.1.5 Game



6.1.5.1 Attributes

`Player player`

The player

`Map map`

The game map

6.1.5.2 Methods

`public Game(StreetTile t, Player c)`

Game constructor

`public void save()`

Saves the game

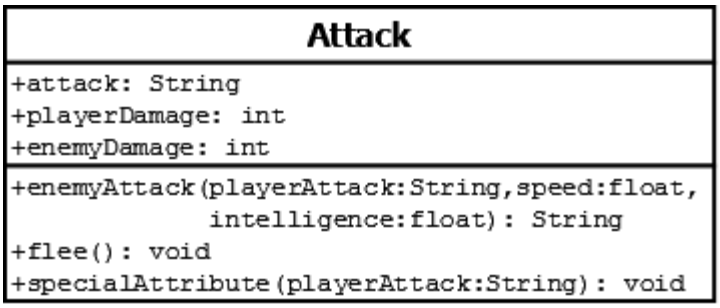
`public void load()`

Loads a saved game

6.1.5.3 Instance Details

There will only be one instance at a time

6.1.6 Attack



6.1.6.1 Attributes

`String attack`

The attack type that the player has chosen

`int playerDamage`

The damage the player sustains, to be subtracted from player.health

`int enemyDamage`

The damage the enemy sustains, to be subtracted from enemy.health

6.1.6.2 Methods

`public String enemyAttack(String attack, float speed, float intelligence)`

Chooses the enemy attack based on probabilities given from speed and intelligence of the player.

Returns a string of the enemy attack type

`public void flee()`

Makes a call to the special message class to display the flee options

`public void specialAttribute(String attack)`

Administers the special attribute of each attack to the enemy

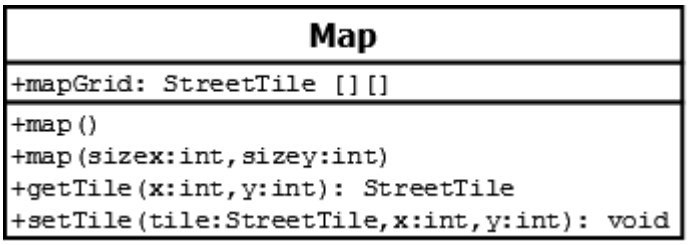
6.1.6.3 Instance Details

One instance at a time for each fight

6.1.6.4 Pseudo Code for the enemy's attack

```
enemyAttack(pizzaBoxSmash, 10, 15)
      translate speed into a probability
      translate intelligence into a probability
      random number between 0 and 2, factoring intelligence
      random boolean to decide if the player dodges, using speed
      if player dodges
            return null
      else
            return attack as string
```

6.1.7 Map



6.1.7.1 Attributes

`StreetTile[][] mapGrid`

Two dimensional array which holds the street tiles containing making up the map

6.1.7.2 Methods

`public Map()`

Default constructor. Calls the main map constructor with default values. (100 x 100)

`public Map(int sizex, int sizey)`

Map Constructor. Generates an array of sizex * sizey of StreetTiles.

`public StreetTile getTile(int x, int y)`

Returns the StreetTile object at the location specified by x and y

`public void setTile(StreetTile tile, int x, int y)`

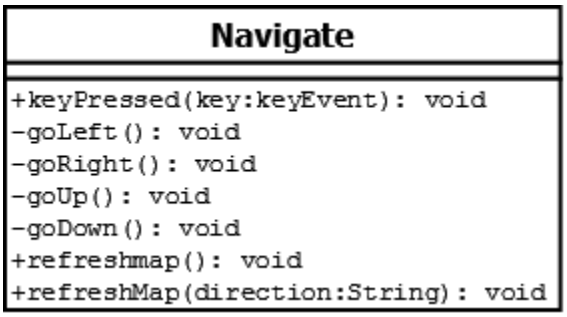
Sets the street tile at x and y with the StreetTile tile argument.

6.1.7.3 Instance Details

There will only be one instance at a time

6.2 Control Package

6.2.1 Navigate Class



6.2.1.1 Attributes

None

6.2.1.2 Methods

`public void keyPressed(keyEvent key)`

Gets the value of key, checking for validity and calling the appropriate method. If left key is pressed, call goLeft(), etc

`private void goLeft()`

Check the tile to the left of the current player. If it is not a wall, set the player's current tile to that tile. If it is a wall, don't move player. Call refresh tile with "left" as an argument.

`private void goRight()`

Check the tile to the right of the current player. If it is not a wall, set the player's current tile to that tile. If it is a wall, don't move player. Call refresh tile with "right" as an argument.

`private void goUp()`

Check the tile up from the current player. If it is not a wall, set the player's current tile to that tile. If it is a wall, don't move player. Call refresh tile with "up" as an argument.

`private void goDown()`

Check the tile down from the current player. If it is not a wall, set the player's current tile to that tile. If it is a wall, don't move player. Call refresh tile with "down" as an argument.

```
public void refreshMap()
```

call refreshMap(“up”)

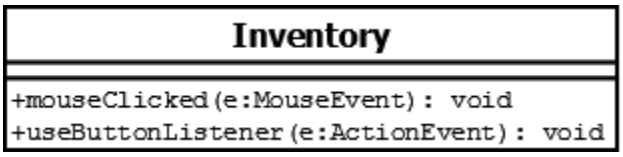
```
public void refreshMap(String direction)
```

Check if there is an enemy on player’s current tile. If there is, call Fight class to change the view to the fight window.
If there isn’t an enemy on that tile, call randomQuality in StreetTile, and regenerate the map graphic around the player’s new location. Generate a sprite image on the center of the map of the player based on direction.

6.2.1.3 Instance Details

This class will only have a single instance at a time.

6.2.2 Inventory Class



6.2.2.1 Attributes

None

6.2.2.2 Methods

```
public void mouseClicked(MouseEvent e)
```

For selecting (highlighting) an item in inventory

```
public void useButtonListener(ActionEvent e)
```

Once an item is selected, press okay to use the item

6.2.2.3 Instance Details

A single instance of inventory is always running

6.2.3 Save Game Class

6.2.3.1 Attributes

String name

6.2.3.2 Pseudocode

```
saveScreen.save.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e)
            add actionListener to save button
            game.save()
      }
saveScreen.back.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e)
            add to back button
            returns to the previous screen
      }
```

6.2.3.3 Instance Details

Instantiated when the save screen is open

6.2.4 Character Creation/New Game Class

6.2.4.1 Attributes

```
String name
boolean sex
```

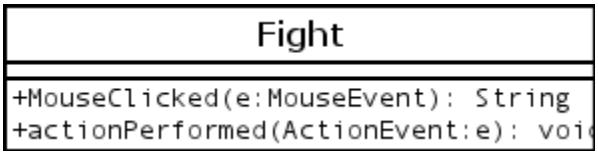
6.2.4.2 Pseudocode

```
openingMenuScreen.continue.addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e)
        add to start button
        calls player constructor
        player(name, endurance, speed, intelligence,
        strength)
}
openingMenuScreen.exit.addActionListener(new
ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e)
        add to back button
        returns to opening menu screen
}
```

6.2.4.3 Instance Details

There will only be one instance of this class at a time

6.2.5 Fight Class



6.2.5.1 Attributes

None

6.2.5.2 Methods

`public String MouseClicked(MouseEvent e)`

Clicking on an attack name displays additional information about that attack type

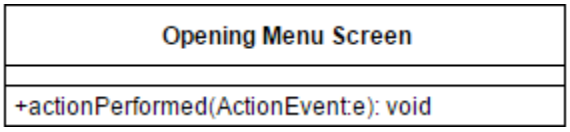
`public void actionPerformed(ActionEvent e)`

To be added to a “Cheese ‘Em” button and holds the calls to make once an attack is selected

6.2.5.3 Instance Details

Instantiated when the fight screen is visible

6.2.6 Opening Menu Class



6.2.6.1 Attributes

None

6.2.6.2 Methods

`public void actionPerformed(ActionEvent e)`

To be added to several buttons to open a new screen corresponding to each button

6.2.6.3 Instance Details

There will only be one instance at a time

6.2.7 High Scores

High Scores
+actionPerformed(ActionEvent:e): void

6.2.7.1 Attributes

None

6.2.7.2 Methods

```
public void actionPerformed(ActionEvent e)
```

To be added to a “confirm” button to return to the “Opening Menu Screen”

6.2.7.3 Instance Details

There will only be one instance at a time

6.3 View Package

6.3.1 Opening Menu Screen

Opening Menu Screen
-mainPanel: JPanel -continue: JButton -newGame: JButton -highScore: JButton -exit: JButton -listen: ButtonListener
+actionPerformed(e:ActionEvent): Void

6.3.1.1 Attributes

```
private JPanel mainPanel
```

The windows main panel

```
private JButton continue
```

The continue game button

`private JButton newGame`

The new game button

`private JButton highScore`

View the high scores

`private JButton exit`

The exit button

`private ButtonListener listen`

The button listener for the buttons

6.3.1.2 Methods

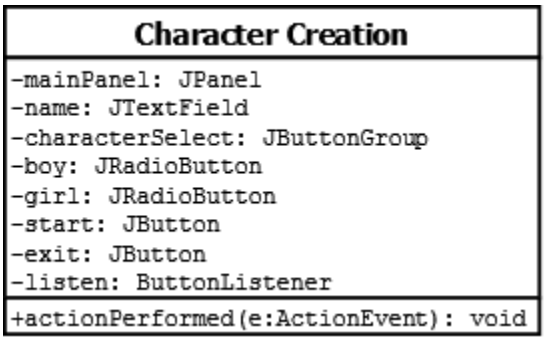
`public void actionPerformed(ActionEvent e)`

Action handler for when a button is clicked

6.3.1.3 Instance Details

There will only be one instance of the title screen at a time

6.3.2 Character Creation Screen



6.3.2.1 Attributes

`private JPanel mainPanel`

The windows main panel

`private JTextField name`

Text field to enter character name

`private JButtonGroup characterSelect`

Group of buttons to select boy or girl character

`private JRadioButton boy`

The boy character button

`private JRadioButton girl`

The girl character button

`private JButton start`

The button starts the game

`private JButton exit`

Cancels everything and returns to the title screen

`private ButtonListener listen`

The button listener for the buttons

6.3.2.2 Methods

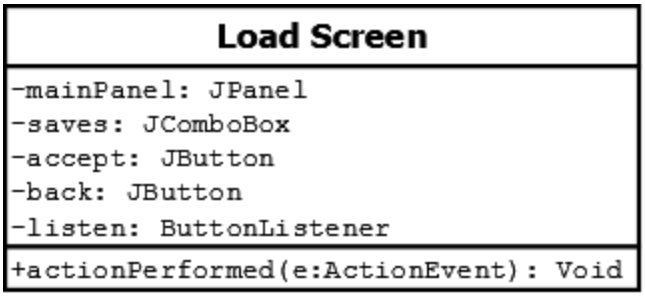
`public void actionPerformed(ActionEvent e)`

Action handler for when a button is clicked

6.3.2.3 Instance Details

There will only be one instance of the title screen at a time

6.3.3 Load Screen



6.3.3.1 Attributes

`private JPanel mainPanel`

The windows main panel

`private JComboBox saves`

A list of possible saves to load

`private JButton accept`

Accepts the selection of a save file and loads it

`private JButton back`

Returns to the previous screen

`private ButtonListener listen`

The button listener for the buttons

6.3.3.2 Methods

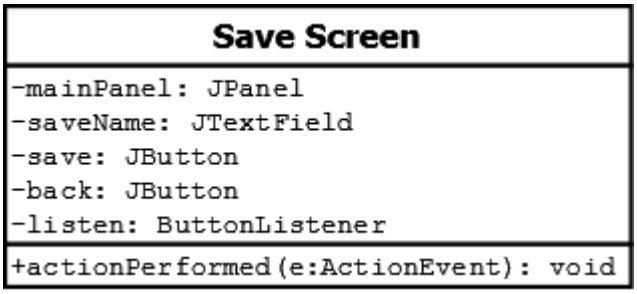
`public void actionPerformed(ActionEvent e)`

Action handler for when a button is clicked

6.3.3.3 Instance Details

There will only be one instance of the title screen at a time

6.3.4 Save Screen



6.3.4.1 Attributes

`private JPanel mainPanel`

The windows main panel

`private JTextField saveName`

User enters a name to give the save file

`private JButton save`

Saves the current game

`private JButton back`

Returns to the previous screen

`private ButtonListener listen`

The button listener for the buttons

6.3.4.2 Methods

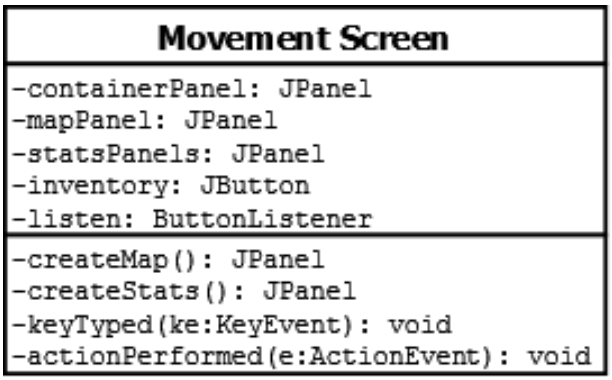
`public void actionPerformed(ActionEvent e)`

Action handler for when a button is clicked

6.3.4.3 Instance Details

There will only be one instance of the title screen at a time

6.3.5 Movement Screen



6.3.5.1 Attributes

`private JPanel containerPanel`

Holds the panels

`private JPanel mapPanel`

The panel that displays the map and current location

`private JPanel statsPanel`

The panel that displays the players stats

`private JButton Inventory`

Opens the players inventory

`private ButtonListener listen`

The button listener for the buttons

6.3.5.2 Methods

`private JPanel createMap()`

Creates the map panel

`private JPanel createStats()`

Creates the stats panel

`private void keyTyped(KeyEvent ke)`

Handles the actions that key presses cause (movement)

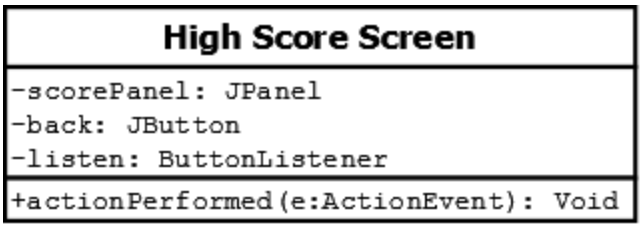
`public void actionPerformed(ActionEvent e)`

Action handler for when a button is clicked

6.3.5.3 Instance Details

There will only be one instance of the title screen at a time

6.3.6 High Score Screen



6.3.6.1 Attributes

`private JPanel scorePanel`

The panel that displays list of high scores

`private JButton back`

Returns to the previous screen

`private ButtonListener listen`

The button listener for the buttons

6.3.6.2 Methods

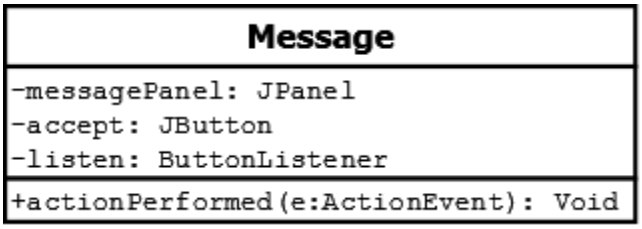
`public void actionPerformed(ActionEvent e)`

Action handler for when a button is clicked

6.3.6.3 Instance Details

There will only be one instance of the title screen at a time

6.3.7 Message Screen



6.3.7.1 Attributes

`private JPanel messagePanel`

Displays a message to the player

`private JButton accept`

Closes the message

`private ButtonListener listen`

The button listener for the buttons

6.3.7.2 Methods

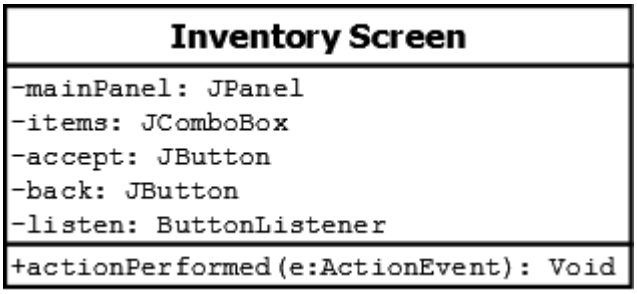
`public void actionPerformed(ActionEvent e)`

Action handler for when a button is clicked

6.3.7.3 Instance Details

There will only be one instance of the title screen at a time

6.3.8 Inventory Screen



6.3.8.1 Attributes

`private JPanel mainPanel`

The windows main panel

`private JComboBox items`

A list of the players current items

`private JButton accept`

Accepts the selected item and applies its effect

`private JButton back`

Returns to the previous screen

`private ButtonListener listen`

The button listener for the buttons

6.3.8.2 Methods

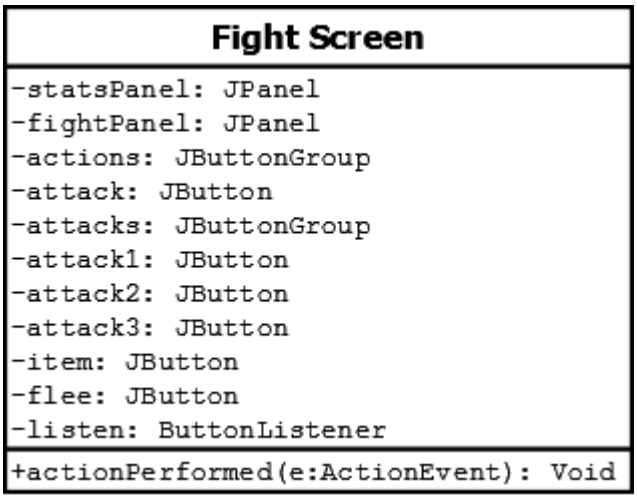
`public void actionPerformed(ActionEvent e)`

Action handler for when a button is clicked

6.3.8.3 Instance Details

There will only be one instance of the title screen at a time

6.3.9 Fight Screen



6.3.9.1 Attributes

`private JPanel statsPanel`

The panel displaying stats

`private JPanel fightPanel`

The panel displaying the fight components

`private JButtonGroup actions`

Group of buttons to select an action

`private JButton attack`

Initiates an attack

`private JButtonGroup attacks`

Group of buttons to select an attack

`private JButton attack1`

Uses attack 1

`private JButton attack2`

Uses attack 2

`private JButton attack3`

Uses attack 3

`private JButton item`

Open inventory to select and use an item

`private JButton flee`

Flees from the fight

`private ButtonListener listen`

The button listener for the buttons

6.3.9.2 Methods

```
public void actionPerformed(ActionEvent e)
```

Action handler for when a button is clicked

6.3.9.3 Instance Details

There will only be one instance of the title screen at a time

7. Implementation Schedule

These dates represent the various stages of the development cycle as our team is allocating work. The dates of completion are the goals in which we hope to accomplish each task, and will usually be before the date listed.

Stage	Date of Completion	Personnel responsible
Design Document Complete	March 8, 2016	Martin Main
Blueprints turned into skeleton code for whole system	March 11, 2016	Martin Main
Iteration 1 - Basic functionality of important classes and methods	March 16, 2016	Martin Main
Iteration 2 - Unit testing classes working, interactions working mostly	March 21, 2016	Martin Main
Iteration 3 - Whole system interacting as expected	March 28, 2016	Martin Main
Alpha 1 - Whole system working without serious bugs, including graphical and audio components	April 1, 2016	Martin Main

System Test Plan

T1. Introduction

The purpose of this document is to provide a complete description of the testing plan and procedures for the system. It is provided as a reference to the development team and as an assurance of quality to our clients. It is divided into four parts, the unit testing, the integration testing, the system testing, and the test scheduling and personnel management.

The purpose of the unit testing is to test the individual classes and subcomponents of the system. The purpose of integration testing is to test the interactions and integration of the various classes and components of the system, and to make sure they all work together as expected. The purpose of system testing is to test a number of use cases on the entire system to make sure everything is running well. This will include testing the downloading and installation of the whole system on different platforms.

Each section will include a test plan, test design, test cases, and test procedures. The purpose of the test plan is to give an overview of what is being tested, the scope of the testing procedures, and the objectives of each section. The test design will describe the approach used in testing each section, and list the tests being made. The test cases will contain a number of actual tests with input values, preconditions and postconditions to be performed on various aspects of the system. The test procedures will describe some of the logistics of setting up tests and performing them on the system.

T2. Unit Test Documentation

T2.1. Introduction

This game is made up of various classes which hold methods and attributes that will be called and used throughout gameplay. These classes will be tested individually using unit tests which tests the functionality of the methods in each class. The tests makes the class perform a set of actions and evaluates the output to determine success.

T2.2. Test Plan

T2.2.1 Items Under Test

All classes of the model, view, and controller layers will be tested upon implementation of the system. This document lists a select few classes that have a large amount of dependencies from other classes.

T2.2.2 Scope

Each class in the model and control layer of Mom and Dad's Pizza Pad will have an associated unit test class. The classes in the view layer will not be tested using unit tests.

T2.2.3 Objectives and Success Criteria

Our objective is to find as many errors as possible in the model and control layers so that dependent classes are not affected indirectly.

T2.3. Test Design

T2.3.1 Approach

The approach of the unit tests will follow a basic outline of testing, reporting, repairing and retesting. The tests will be performed according to the test cases given, and reports will be generated using the sheets provided in Appendix A. The reports will be given to the programmers who have written the class in question, and they will be repaired. Once repairs are complete, the programmer will informally test for the given bug, and retesting of the class as a whole will commence.

T2.3.2 Items Tested

- Use attack
- Use Item
- Character Creation

T2.4. Test Cases

T2.4.1 Use Attack

Class: Attack
Purpose: Handle the calculations involved with attacking an enemy
Precondition: The player is in the fight screen, and there is an instance of an enemy
Input:

playerAttack = pCutter;
speed = 10
intelligence = 15

Postconditions: The enemy can choose a number of attacks, based on probability. The probability that the enemy will choose pizza box smash as his attack will be highest, since the player has an intelligence > 0

T2.4.2 Use Item

Class: Items
Purpose: Alter the player attributes, depending on the item chosen
Precondition: The player has more than 0 of the requested item
Input:

name = soda

Postconditions: call the method applyEffect(soda), which in turn calls applySoda(). This method will increment the players energy level using getters and setters and also decrement the soda quantity by 1.

T2.4.3 Character Creation

Class: CharacterCreation
Purpose: To initialize a new players attributes
Preconditions: A name has been entered
Input:

name = Jerry
endurance = 20
speed = 10
intelligence = 10
sex = 0 //*female*

Postconditions: Use getters of the player class to confirm that the character has attributes that match the input.

T2.5. Test Procedures

Write main functions for the necessary classes being tested and in the main functions write the code necessary to run the tests outlined in the previous test case section. Results will be recorded in the test form (Appendix A1). For the unit tests, we will aim to have our error count below 20 for the whole code base before moving on to integration testing, and below 10 before moving on into system testing. We will aim to have below 5 observable errors of low importance at the beta release of the system.

T3. Integration Test Documentation

T3.1. Introduction

During the development of the game it will go through a series of versions. Each version will add more and more classes and functionality to the game. The purpose of integration testing is to test the interactions and integration of the various classes and components of the system, and to make sure they all work together as expected throughout each iteration of development.

T3.2. Test Plan

T3.2.1 Items Under Test

The integration tests will test the component that have the most complex relationships with each other, and test out how those classes and methods communicate and interface under many conditions.

T3.2.2 Scope

The integration tests will include a sample of tests of interaction between components, including the input and output states of the system, and the steps taken to produce the interaction.

T3.2.3 Objectives and Success Criteria

The objectives of the integration tests is to discover any errors in the code or logic of the classes and their interactions with each other. It is aimed at testing for the expected output of the system when different classes interact in complex ways.

T3.3. Test Design

T3.3.1 Approach

The approach of the integration tests will follow a basic outline of testing, reporting, repairing and retesting. Integration tests will use a more general testing strategy than unit tests, but less flexible than system testing.

T3.3.2 Items Tested

Test 1: Use Mr. Michel's Map Item
Test 2: View-Control interactions

T3.4. Test Cases

T3.4.1 Use Mr. Michel's Map Item

Purpose: Test all the functionality of how the inventory screen interacts with the items class, and how that interacts with the map and map screen classes.
Precondition: The map doesn't show very much area nearby.
Steps: Click on the map item in inventory, and press OK button.
Postconditions: The inventory screen should disappear, and immediately the map should display a larger field of view, including the locations of any hidden objects and enemies.

T3.4.2 View-Control Interactions

Purpose: Test how the screens in the game show up and display information.
Steps: From menu screen, click on every button and back buttons, then in game click on every button, then return to menu and repeat 2 times. Enter battle screen, and test all buttons on that screen.
Expected Result: Every button click should initiate an immediate result, and screens should be switched to the next screen as defined in the detailed design. The content of the screen should be consistent, but this test only requires the screens to switch promptly and correctly.

T3.5. Test Procedures

The integration tester will need a working build of the system to manually carry out the test cases described above. The tester will get a list of test cases to be tested, and fill out the test report document in Appendix A. This document will be delivered to the head of the development team to assign to whoever they deem suitable. We aim to ensure integration errors are at a minimum before working on system testing.

T4. System Test Documentation

T4.1. Introduction

System tests will test whether the architecture of the design has been correctly implemented. This will ensure the Model, View and Control packages are all working together properly. The completion of these tests will ensure the system is integrated correctly.

T4.2. Test Plan

T4.2.1 Items Under Test

The items being tested will show that the games system works properly. This will include different packages working with one another through changing screens and moving data from package to package.

T4.2.2 Scope

The scope of these tests cover the interaction of the Model, View and Control packages.

T4.2.3 Objectives and Success Criteria

These tests will be used to find problems within the system. If a problem is found it will be sent back to be corrected. After correction the test will be repeated. This will happen until the test is passed.

T4.3. Test Design

T4.3.1 Approach

The approach of the unit tests will follow a basic outline of testing, reporting, repairing and retesting.

T4.3.2 Items Tested

Below is an example of how a test will be performed on the Street Tile class as well as downloading and running the game on a system.

T4.4. Test Cases

4.4.1 Street Tile Class

Test Number	Test Description	Steps	Pre Conditions	Post Conditions
4.4.1 T1	Attempt to change tile	While on a tile select a direction to move	Player is on a tile with a tile adjacent	If valid the window will change to the street tile in the selected direction
4.4.1 T2	Check if the selected direction is valid	Player selects a direction to move	Player is on a tile with at least one adjacent tile that can be moved to	Game checks if the selected direction is a street(valid move) or a wall(invalid move) if valid the player progresses to the new tile
4.4.1 T3	Initiate a battle when entering a new tile	Enter a new tile that hasn't been previously visited and has an enemy	Player is on a tile with at least one adjacent tile that can be moved to The adjacent tile hasn't been previously visited The enemy boolean is true for the destination tile	The enemy boolean will be true therefore a battle will start.
4.4.1 T4	Receive an item in a new tile	Enter a new tile that hasn't been	Player is on a tile with at least one	The enemy boolean will be

		visited and doesn't have an enemy.	adjacent tile that can be moved to The adjacent tile hasn't been previously visited The enemy boolean is false for the destination tile	false therefore a randomizer will determine if the player will get an item and what item it is

4.4.2 Downloading and Running the Game

Test Number	Test Description	Steps	Pre Conditions	Post Conditions
4.4.2 T1	Downloading the game	Click on the file on the website to download it. Attempt this on multiple operating systems(Windows, Mac, Linux)	Computer with an internet connection and enough free hard drive space to hold the game.	Game downloads correctly
4.4.2 T2	Launching the game	Click and run the downloaded file, attempt on multiple operating systems(Windows, Mac, Linux)	Game file downloaded	Game launches succesfully and is playable
4.4.2 T3	Game is playable	Attempt to create a character and play a bit of the game.	Downloaded and running game	The game can be played on the system

T4.5. Test Procedures

The tester will load the necessary source files and test files and run them. The tester will follow the steps outlined in the test cases in the previous section filling out the test form (Appendix A1) with the results from the test.

T5 Schedule and Personnel

The scheduling of the testing will be have significant overlap, because of the nature of the project and the testing procedures being used. We expect Unit testing to continue somewhat as we move into integration and then system testing, although it will be the most near the beginning. We will limit the full system testing until the unit test and integration test stages are coming to a close.

Testing Period	Dates of testing	Personnel responsible
Unit Testing	March 14-24, 2016	Scott Barnett
Integration Testing	March 18-30, 2016	Scott Barnett
System Testing	March 24-April 4, 2016	Scott Barnett

Appendix A

A1. Example Test Form

Tester Name:		Test number:	
Class:			
Description:			
Input Values/Input State:			
Output Values/ Output State:			
Result: (Circle one)	Error		Success
Severity:	Low	Medium	High	Signature:
Description of error:			

A2. Example Test for Items

<table>
<tr><td>Name:</td><td>Jimmy</td><td>Test number:</td><td>1.0</td></tr>
<tr><td>Class:</td><td colspan="3">Items</td></tr>
<tr><td colspan="4">Description: Edit Items effect on Players attributes</td></tr>
<tr><td colspan="4">Input Values/Input State: The name of the item as a string, and the current state of the players attributes.</td></tr>
<tr><td colspan="4">Output Values/ Output State: Soda did not affect player’s energy as expected. All other methods worked as expected.</td></tr>
<tr><td>Result: (Circle one)</td><td colspan="2">Error</td><td>Success</td></tr>
<tr><td>Severity:</td><td>Low</td><td>Medium</td><td>High</td><td rowspan="2">Signature:
Jimmy Flemming</td></tr>
<tr><td>Description of error:</td><td colspan="3">If there is an error, the item use will not reflect in the player’s attributes. This error is not game-breaking, but it could be very frustrating for the user.</td></tr>
</table>