Joel Penney, Paul Rich, Jeffrey Thompson, Zach Van Dyke, Luke Penney, Drew Smith
May 3

Final Project Report

Updated requirements

Navigate Inventory

Keep track of page:
System must keep track of where in the menus the user is
Keep track of selected options:
System must maintain the states of the options and their effects
What to do when an option is selected:
System must enact the effects of option selection

Navigate overworld

Pathfinding:
The way that NPCs move around the overworld
AI movement and actions:
The movement patterns that NPCs have and the actions that take place when the NPCs interact with other objects
Handling player movement:
Actions that happen based on player movement
Measuring objectives, handling puzzles:
Knowing how an objective/puzzle is to be completed, and understanding how close it is to completion
Zone transitioning:
Handling what happens when the player/NPCs move into a new area

Encounter System

Offense:
Recognize attacks from both the player and AI, and apply the proper effects and damage values associated with them.
Defense:
Recognize defensive actions from the player and AI, and apply the proper effects.
Stats management:
Handle buffs and debuffs created through character and AI moves, as well as initial values attained at the commencement of the encounter.
Inputs:
Handle inputs including but not limited to swipes and multi-pointed movements for character actions, as well as handling the timing of said inputs in relation to the specified timings in the game.

Item management

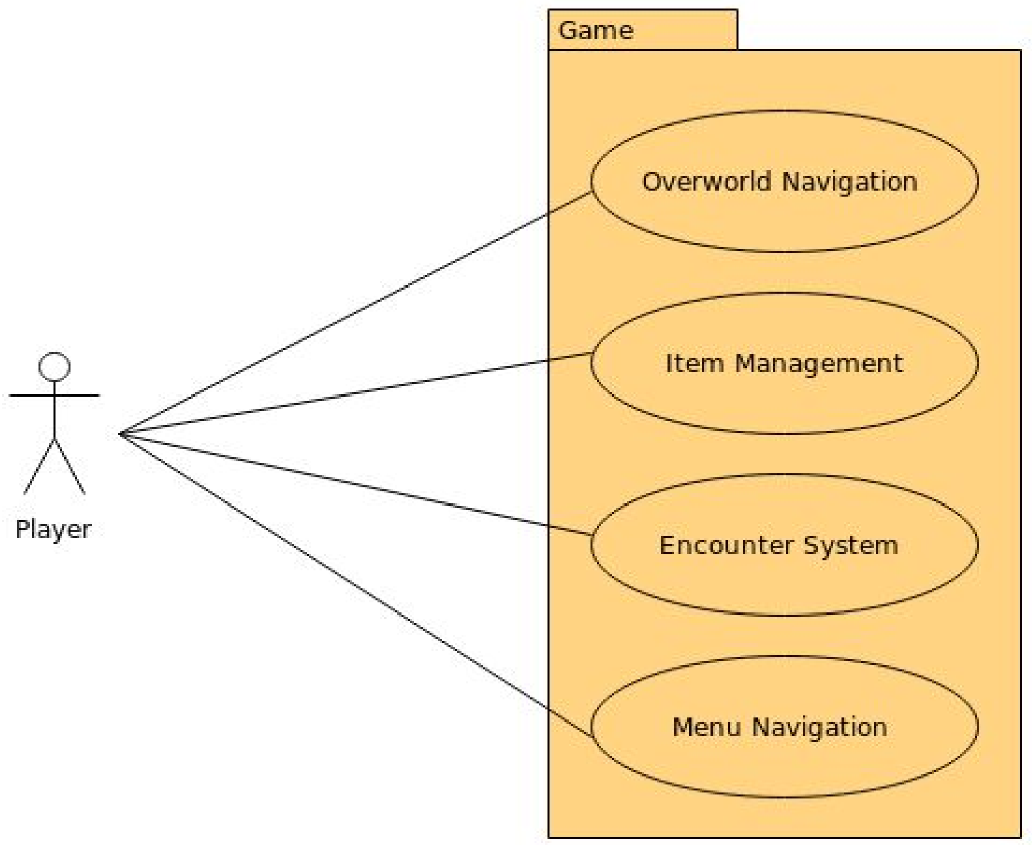
Add to inventory:

Take items from the overworld and add them to the users inventory
Delete/drop items:
Allow for a player to both drop and delete items (depending on choice).
Whether items complete tasks:
Understand and handle circumstances in which item attainment results in quest or objective completion.
Using items/item effects:
Recognize and perform the appropriate operation when items are used. Properly affect character stats when items offer buffs to attack, defense, or other stats.

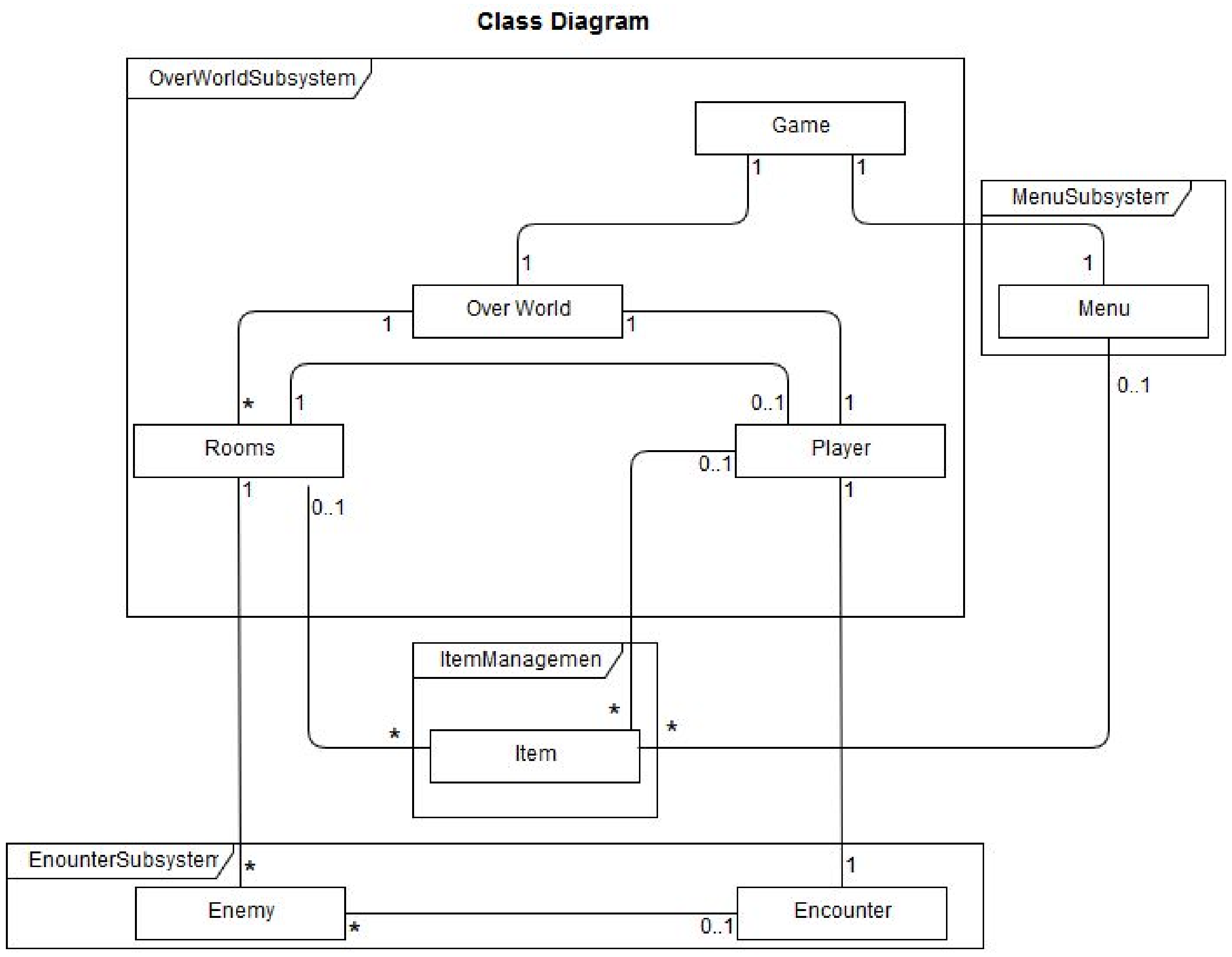
Non-functional/domain requirements
Android platform
Fast responses to player inputs
Fun

Updated use cases, class, sequence, and deployment diagrams

#5. Top level use case diagram

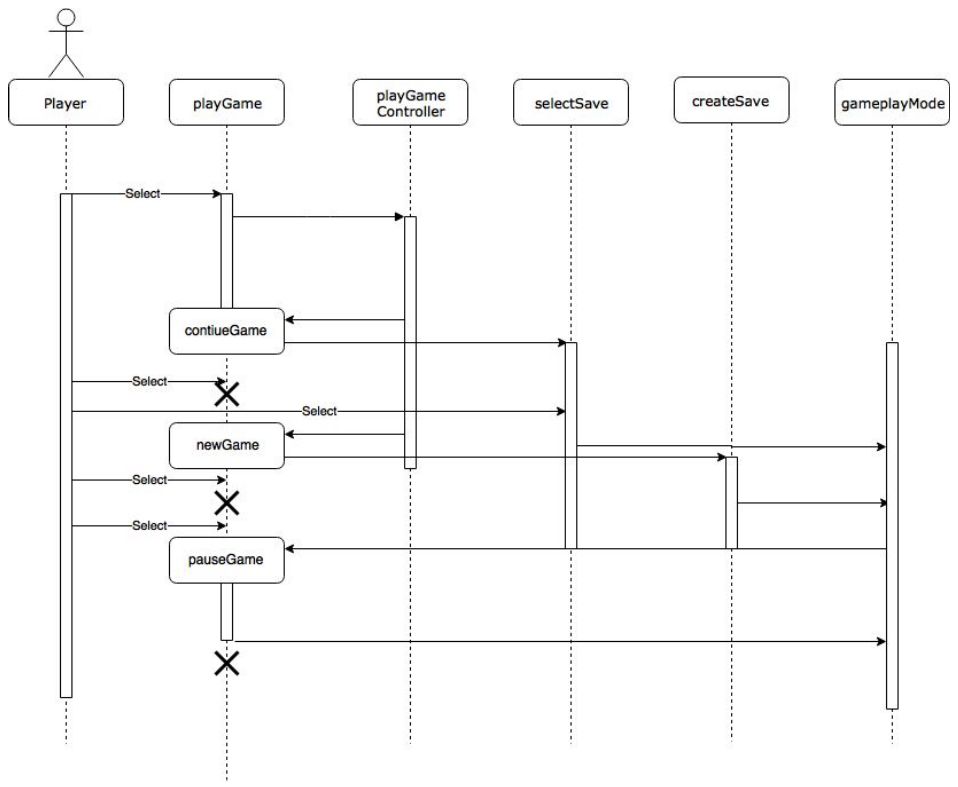


#6.Class Diagram

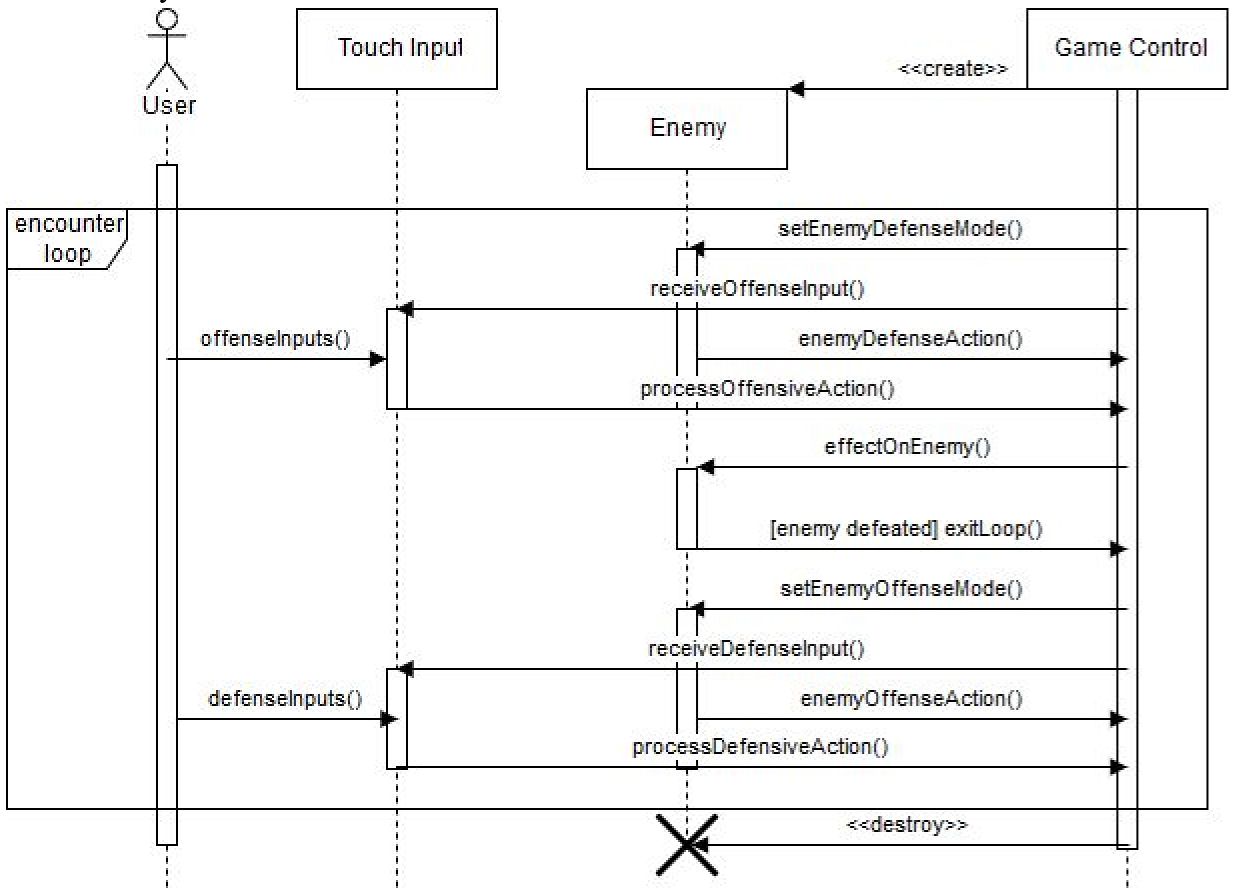


#7. Sequence Diagrams

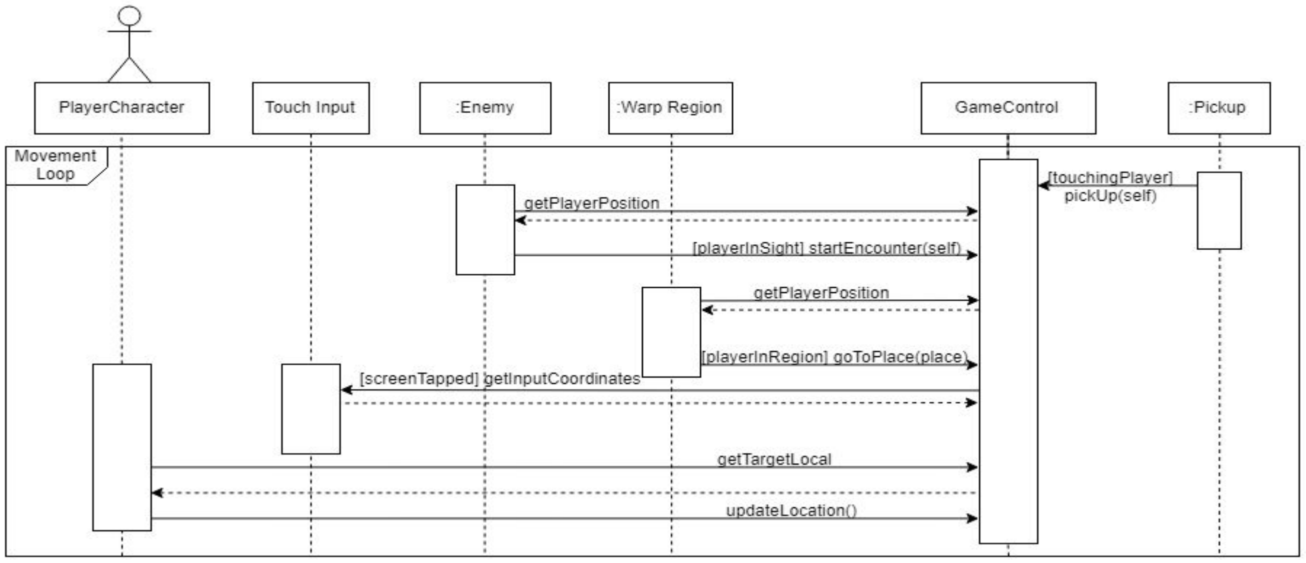
Menu Navigation



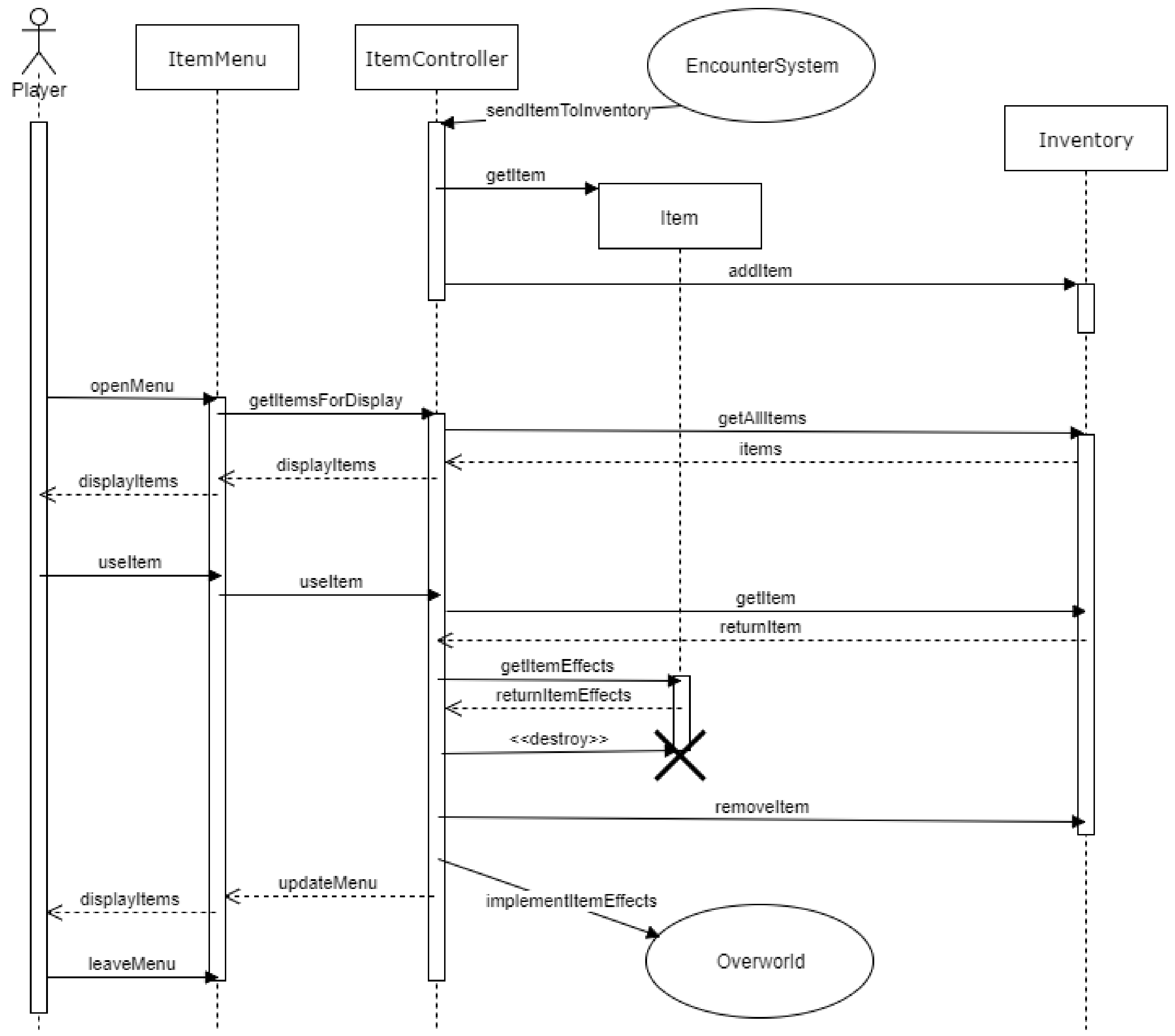
Encounter System



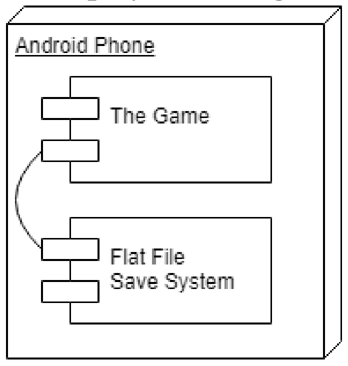
Overworld Navigation



Item Management



#8. Deployment Diagram

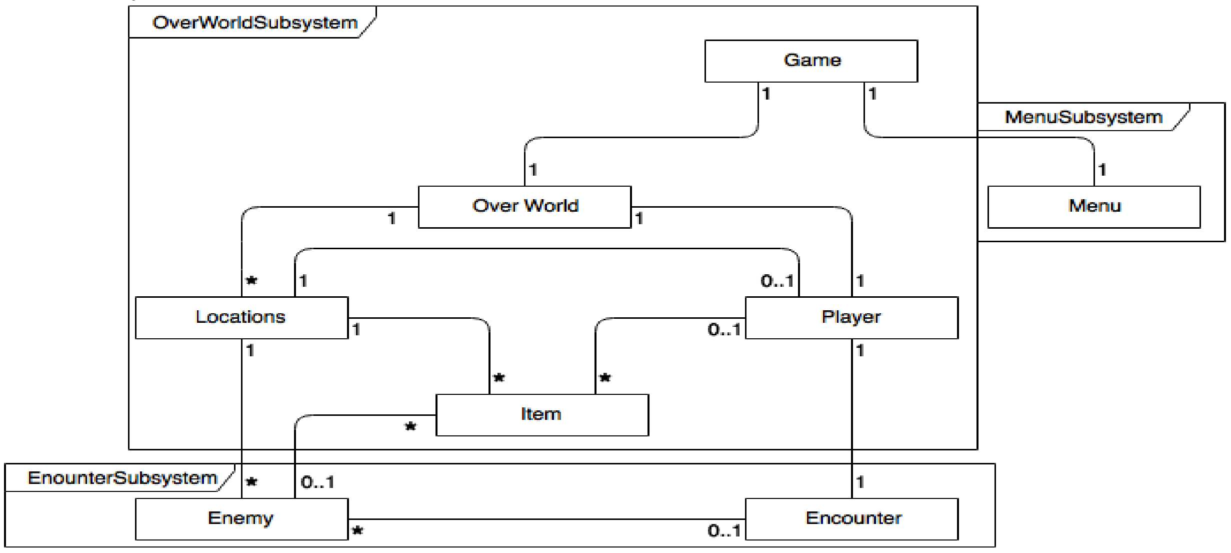


#9. Flat File Schema

Field Name	Data Type	Field Size	Description
Player Info	**Singleton**		**Information pertaining to player progression**
Level Info	BitVector	16	Level info for the player
Inventory Info	BitVector	32	Information on items and gear collected
Equipped Info	BitVector	16	Information on items and gear equipped
Level Info	**Array**		**Information pertaining to Level or Room completion**
Lamp Info	BitVector	16	Bitvector with direct correlation to an array of lamps
Baddy Info	BitVector	16	Bitvector with direct correlation to an array of baddies
Item Info	BitVector	16	Bitvector with direct correlation to an array of items
Other/Misc Info	**Singleton**		**Any info not directly related to the Player or Level**
Player Position	Serialized Ve	~	The physical coordinates of the Player-character
Loaded Scene	String	16	The scene loaded when saved

Due to only needing to track individual save-files, a flat-file system was found to be the simplest database to implement that would also suit our needs.

#10. Subsystems and their role



Updated written specifications

Overworld Navigation

1. Navigation begins: PlayerCharacter, Enemies, WarpRegions(Doors), and Pickups(Items) are loaded into the area and the movement loop is initiated.
    2. PlayerCharacter is given any Pickups it is currently colliding with, movement loop continues.
    3. Enemies and WarpRegions check if PlayerCharacter is within their regions of interaction
        4. Enemies initiate encounter if so, WarpRegions send player to new area (both end movement loop).
    5. PlayerCharacter's target position moves to position tapped on screen (or last position tapped).
        6. PlayerCharacter moves towards target position.
    7. Movement loop repeats as necessary.
8. Encounter begins or Warp activated.

Item Management

1. Item is added to the inventory
    2. The game manager gives an item to the inventory
    3. The inventory checks whether it contains other items of the same type. If so, it increments the number of items of it has of that type; if not, it adds that item with a quantity of one.
4. Player uses item in the inventory
    5. The user opens the items menu. The items that are in the inventory are displayed, along with the quantities of each.
    6. The user selects an item and uses it.
        7. When used, the item causes some effect to happen (i.e., a key may open a door)
        8. The inventory reduces the quantity of that item by one, or if there was only one, deletes it from the storage

Encounter System

1. Encounter begins: game creates enemy entity and initiates encounter loop
    2. Game sets enemy on defense mode and player on offense mode
        3. Player makes offensive action, enemy makes defensive action
    4. Game calculates the effect on the enemy
    5. Game sets enemy on offense mode and player on defense mode
        6. Player makes defensive action, enemy makes offensive action
    7. Game calculates effect on player
    8. Encounter loop repeats as necessary
9. Enemy defeated; encounter ends. Game removes enemy object

Menu Navigation

1. Player selects button "Play Game"
    2. System presents options of "Continue Game" and "New Game"
    3. Player selects "Continue Game"
        4. System presents list of Saves to choose from
        5. Player selects a Save
        6. Player enters gameplay
    7. Player selects "New Game"
        8. A fresh Save is created for the Player
        9. Player enters gameplay
    10. Player selects pause
        11. Gameplay freezes until unpaused

A list of subsystems and their roles

Overworld Subsystem Interface

```
interface Overworld {
// deals with movement and interactions in the rooms
        void EnterEncounter (string scene);
        void AddItem (string item);
}
```

Inventory Subsystem Interface

```
interface Inventory {
// defines the item management subsystem
        void openItemMenu();
        void addItem(Item i);
        void useItem(Item i);
        void resumeGame();
}
```

Menu Subsystem Interface

```
interface Menu {
// allows users to start a new game or resume a saved game
        void NewGame();
        void ResumeGame();
}
```

Encounter Subsystem Interface

```
interface EncounterControl {
// defines passing control to the encounter subsystem (subsystem chooses when to exit)
        void Start();
}
```

```
interface DefAction {
// translates user input into character action, and how collisions affect the character
        void Update(Touch[] t); // Unity-provided function, used to take player inputs as part of
defensive phase
        void onCollision(Collision col);
}
```

APIs

Note for how certain methods work in Unity: Monobehaviour methods (Update, Start, OnEnable, FixedUpdate, Awake) are technically private but are called by the system and so we have included them in the subsystem interfaces when relevant (when they act as methods used by other code).

Interface for Encounter Subsystem (includes EncounterControl, Defense, DefAction, and Attack files)

```
/**
* Regulates phase changes and exiting combat in
* the encounter system
*/
public class EncounterControl {
   /* *Prefab for the enemy that the player is encountering  */
   public static GameObject enemyPrefab;

   /* * References to various scripts, so they can be enable or disabled for phase changes */
   private Defense defScript;
   private DefAction defActScript;
   private Attack attackScript;

   /* *  Reference to the player character */
   public GameObject player;

  /* * Initialization of scripts, begin in player offense mode. */
   private void Start ()

    /* This toggles the active offensive and
     * defensive scripts when a phase has completed
     */

   /* *  Update is called once per frame */
  private  void Update ()
}

/** Controls the player's actions during the defensive phase */
public class DefAction {
        /** Initialization; gets references to various objects */
        private void Start ()

        /** Reset tracked finger inputs */
        private void OnEnable ()

        /** Simulates a weaker gravity to increase jump time (each physics frame) */
        private void FixedUpdate()

        /** Take player input */
        private void Update ()
}

/** Creates enemy moves for defense phase */
public class Defense {

        /**
         * Creates enemy moves
         * @invariant attacks is initialized
         */
        private void Update ()

        /** Initialization; gets list of enemy moves */
        private void OnEnable ()

        /** Keeps track of whether the defensive phase has ended */
        public bool Finished()
}

/** Creates circles in patterns for player to trace in order to lower Enemy's Enthusiasm*/
public class Attack {

  /** How long in seconds that the player has to trace each circle before they disappear*/
  public float time = 3;

   /** The text that displays the Enemy's Enthusiasm and Player's Confidence*/
  public Text E;
  public Text t;

   /** Initializes variables*/
  private void Start ()

   /** Called by EncounterControl to check if Attack phase is over*/
  public bool Finished()

   /** Handles a circle being touched by the cursor*/
  public void CircleTouched(GameObject x)

   /** Takes player input*/
  private void Update ()
}
```

Interface for Game Control Subsystem

```
/* *
  * Roughly speaking, this is our "Control" or "Logic" class. It
  * determines what stays alive on scene transition, facilitates
  * saving and loading, and maintains a soft save (or cache)
  * for transitioning between different scenes.
  * /
public class GameControl {

/* *
 * To be used at game instantiation
 */
            /* *  The instance of GameControl representing the current game */
            public static GameControl control;
            /* *  The player instance */
            private GameObject player;

            /* * A reference to the player's prefab in case the player needs to be created */
            public GameObject playerPrefab;

/* *
 * To be used at scene transition(s)
 */
            /* * The array of all scene names, used to transition between scenes */
            public static string[] scenes = new string[]{"TitleScreen","sample","Room01"};

            /* * The int value corresponding to the scene entrance to spawn in
             * on scene load (if a door was used)
             */
            private int door = 0;

            /* * The index of the baddie in the scene whom would die upon return from a successful
              *    encounter
              */
            private int baddieToDie = -1;

            /* *  The position to set the player to on scene load */
            private Vector3? playerPosition = null;
```

```
        /* *  Represents whether or not to load the scene as from a door’s spawn location */
        private bool doorSpawn = false;

/* *
 * To be used in saving/loading
 */
        /* * An array of LevelData classes for storing the data of each level */
        private LevelData[] levels = new LevelData[12];

        /* * The scene the game is currently in */
        public string currentScene;

        /* * How much confidence (which is basically health) the player has */
        private int confidence;

        /* * The maximum amount of confidence the player can have */
        private int maxConfidence = 100;


        /**
         * Initialization:Sets the player's confidence to the max
         */
        public GameControl()


        /**
         * Adds to/removes from the confidence the player has (doesn't allow confidence to go over the
         * maximum, and ends the game if it goes under 0)
         */
        public void AdjustConfidenceBy()

        /**
         * Return's the player's confidence
         */
        public int Confidence(int confidenceChange)


        /**
         * ensures no LevelData datatypes are undeclared (for saving purposes)
         */
        void CreateEmptyLevels()

        /**
         * Further Initialisation:
         * Blanks levelData (to ensure no null reference errors)
         * Ensures that the GameControl and Player classes are singletons within the scene
         */
        void Awake ()
```

```
/**
 * Returns the player object (for use by scenes and scene-restricted objects)
 */
public GameObject GetPlayer()

/**
 * Displays Player’s current confidence level
 */
private void OnGUI()

/**
 * Opens a file stream, serializes necessary information in a binary format, closes file stream
 */
public void Save()

/**
   * Opens a file stream, deserializes necessary information from a binary format and overwrites
   * necessary data, closes file stream
   */
public void Load()

/**
 * Swaps Scenes, for use when entering doors between two overworld scenes
 */
public void SwapScene(int sceneToLoad,int doorToLoad)

/**
 * Enter an encounter scene, with the correct baddie, while keeping track of the level's state
 */
public void EnterEncounter (GameObject baddie)

/**
 * Exit the encounter, spawning in the location from whence the encounter was initiated
 * This will presumably only be called after a successful encounter has ended
 */
public void ExitEncounter ()

/**
 * Determines where the player should spawn in based on whether or not it entered
 * via a door, loading a save, or exiting an encounter successfully.
 */
public Vector3 GetPlayerSpawn()

/**
 * Puts the data for the current level in the levels array, but does not save it into a flat file.
 */
public LevelData[] CacheLevelData()
```

```
	/**
	 * Returns the LevelData from levels of the current scene.
	 */
	public LevelData GetLevelCache()
```

Interface for Menu / Inventory Subsystem

```
/**
 * A sample Item for testing purposes
 */
public class SampleItem
{
	/**
	 * Initialization
	 */
	public void Start ()

	/**
	 * Creates a SampleItem near position (0,0,0)
	 */
	protected override void UseAction ()
}



/**
 * Class for a visual inventory button on the player's screen
 */
public class ButtonIntermediate {

	/**
	* When the button is pressed, open up the inventory menu
	*/
	public void GoClick()
}



/**
 * A script for items that restore confidence
 */
public class ConfidenceItem
{
	/**
	 * The name of this item
	 */
	public string itemsName;

	/**
	 * How much confidence is gained when the item is used
	 */
	public int healthRegained;

	/**
	 * Initialization
	 */
	public void Start ()

	/**
	 * Use the ConfidenceItem - increases the player's confidence
	 */
	protected override void UseAction ()
}



/**
 * A class for holding all the player's items
 */
public class Inventory {
	/**
	* A prefab for the items menu
	*/
	public Transform itemsMenuPrefab;

	/**
	 * Holds the items
	 */
	private List<Item> items;

	/**
	 * Initialization
	 */
	public void Start()

	/**
```

```
         * Add an item to the inventory
         */
        public void addItem(Item newItem)

        /**
         * Get an Item from the items List
         */
        private Item GetItemFromList(Item newItem)

        /**
         * Pauses the game and opens the menu. This should
         * be called by a button or something in the game
         */
        public void OpenMenu()
}

/**
 * An interface for items that can be picked up,
 * put in the inventory, and used by the player
 */
public abstract class Item
{
        /**
         * How many of this item are in this location
         */
        public int quantity;

        /**
         * The name of this kind of Item
         */
        protected string myName;

        /**
         * Returns the name of this Item
         */
        public string Name ()

        /**
         * Returns the number of items of this type that are here
         */
        public int GetQuantity (){

        /**
         * Returns the number of items of this type that are here
         */
        public void SetQuantity (int newQuantity)
```

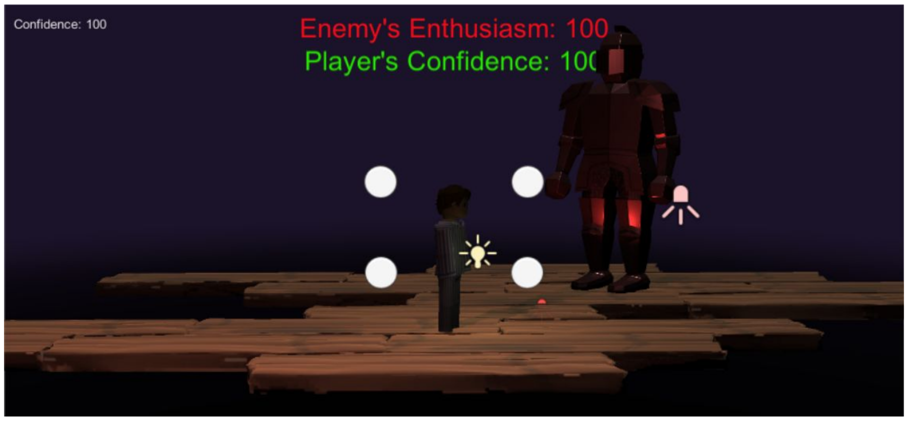
```
            /**
             * If an Item object is lying around and the player
             * collides with it, remove it from the overworld and
             * add it to the inventory
             */
             private void OnCollisionEnter (Collision col)

            /**
             * Use this Item
             */
            public void UseItem ()

            /**
             * What happens when this Item is used
             */
            abstract protected void UseAction ()
}


/**
 * A class to display and control the items menu
 */
public class ItemsMenu
{
            /**
             * A prefab for buttons in the inventory
             */
    public Transform useItemButton;

            /**
             * Stores a reference to the inventory list so it can be updated as needed
             */
            private List<Item> items;

            /**
             * Called when the inventory menu is first opened
             */
    public void StartInventoryMenu(List<Item> inv)

            /**
             * Sets the on-screen menu to match what's in the inventory
             */
            public void UpdateInventoryMenu()

            /**
             * Resume the game. Should be called by a button being pressed.
             */
    public void ResumeGame()
}
```

For each subsystem, screenshots of the main pages that show their functionality

EncounterControl Subsystem

(Offensive phase)

Touch Screen: player touches the screen to toggle each circle as "touched"



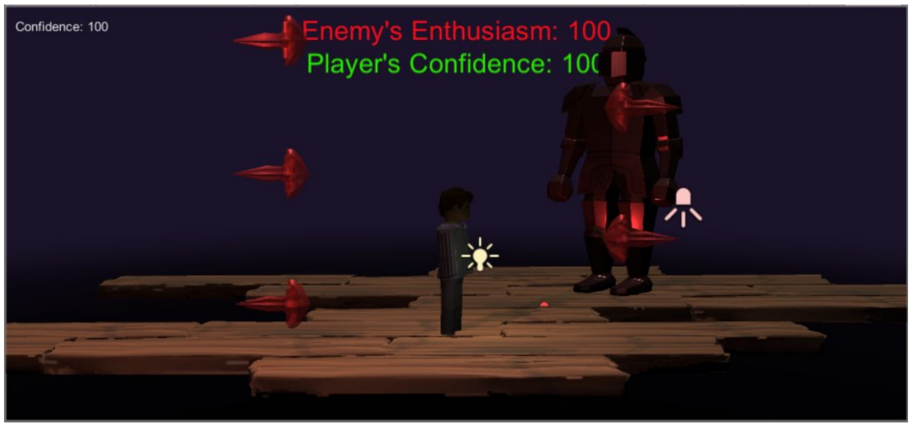
(Defensive phase)

Touch Screen: player swipes the screen to input moves

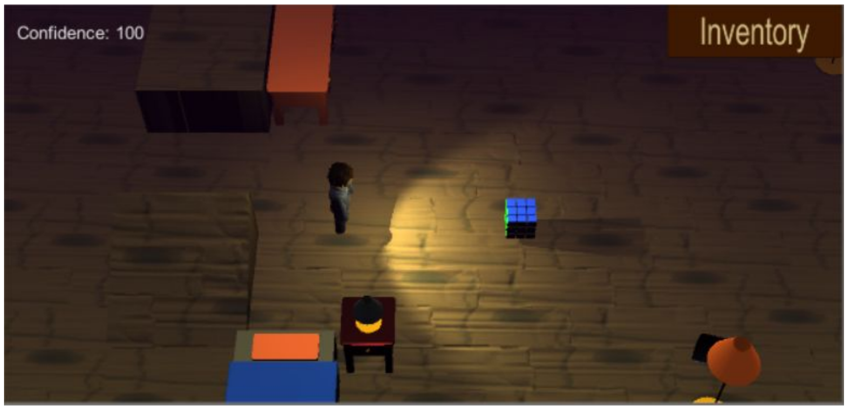
Swipe up: make character jump

Swipe down: make character's shield

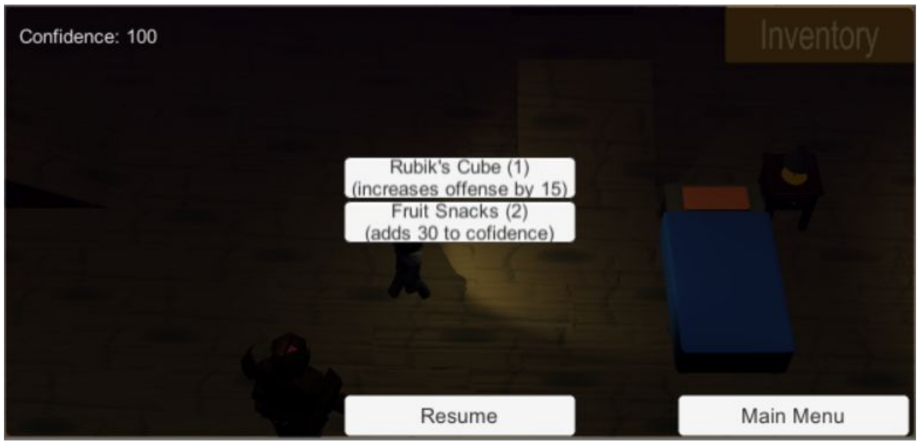
Swipe sideways: make character's blast object



Inventory Subsystem



From the Overworld, the player can pick up items by running into them and open the inventory menu by pressing the inventory button.


Confidence: 100
Inventory
Rubik's Cube (1) (increases offense by 15)
Fruit Snacks (2) (adds 30 to cofidence)
Resume
Main Menu

In the inventory menu, the player can resume the game or exit to the main menu by pressing the relevant buttons. They can also press the button of an item to use that item.