Iteration 1 Source Code

Team Information:

- Team Number: 13
- Group Members:
  - Mitchell Hird (Student 100881078)
  - Albert Cho (Student 100857664)
  - Anthony Tasca (100410783)

Before Reading This Section:

- If you wish to browse code one file at a time you can find the full source code in Source Code folder that is packaged within the zip file this comes in. The source code can be found under the src folder
- Classes will be broken up by package
- Blank classes have been omitted as they are simply place holders that will be expanded upon come next iteration

Package: Apps

Class: AppMagicRealm.java

```
package app;

import views.GameView;

/**
 * Main Class For The Magic Realm Game. Creates A Game View And Starts Everything
 * @author Mitchell
 */
public class AppMagicRealm {
	// Main Method To Launch The Java App
	public static void main (String[] args) {
		new GameView();
	}
}
```

Package: Controller

Class: AppMagicRealm.java

```
package controller;

import java.util.ArrayList;

import views.GameView;
import models.BoardModels.Clearing;
import models.characterModels.PlayerBase;
import models.characterModels.playerEnums.CharacterClass;

public class clientController {
```

```
	private GameView parent;
	private int currentPlayerIndex;
	private PlayerBase currentPlayer;
	private ArrayList<PlayerBase> thePlayers;

	private boolean gameStarted;

	public clientController(GameView theView){
		parent = theView;
		currentPlayerIndex = 0;
		thePlayers = new ArrayList<PlayerBase>();
	}

	// Adds The Player to The Game
	public void addPlayer(CharacterClass playerClass, String playerName){
		// Add The Player Into The List
		PlayerBase aPlayer = new PlayerBase(playerName, playerClass);
		thePlayers.add(aPlayer);

		// If That Is The First Player, Set The Current Player To That
		if (thePlayers.size() == 1) {
			currentPlayer = aPlayer;
		}
	}

	// Returns The player List From The Game
	public ArrayList<PlayerBase> getPlayers(){
		return thePlayers;
	}

	// Moves To Next Player's Turn, Using Modulo Math
	public void moveToNextPlayer () {
		currentPlayerIndex++;
		currentPlayer = thePlayers.get(currentPlayerIndex % thePlayers.size());
		currentPlayer.startPlayerTurn();
	}

	// Starts The Game When Called
	public void startGame () {
		gameStarted = true;
		for (PlayerBase p: thePlayers) {
			Clearing playerStart =
parent.getBoardView().getDefaultClearingForClass(p.getPlayerClass());
			p.setCurrentClearing(playerStart);
			p.setHomeClearing(playerStart);
			playerStart.playerMovedToThis(p);
		}
	}
```

```
	// Remove The Player From The List
	public void removePlayer(String playerToRemoveName) {
		for(int i = 0; i < thePlayers.size(); i++){
			if(thePlayers.get(i).getName() == playerToRemoveName){
				thePlayers.remove(i);
				return;
			}
		}
	}

	// Get Player By Name
	public PlayerBase getPlayer (String name, CharacterClass charClass) {
		for (PlayerBase p: thePlayers) {
			if (p.getName().equals(name) && p.getPlayerClass() == charClass) {
				return p;
			}
		}
		return null;
	}

	// Gather All Of The Unhidden Players
	public ArrayList<PlayerBase> getUnhiddenPlayers () {
		ArrayList<PlayerBase> returnVal = new ArrayList<>();
		for (PlayerBase p: thePlayers) {
			if (!p.isHidden()) {
				returnVal.add(p);
			}
		}
		return returnVal;
	}

	/*----------------  Getters And Setters ------------------------*/
	public PlayerBase getCurrentPlayer() {
		return currentPlayer;
	}

	public boolean isGameStarted() {
		return gameStarted;
	}

	public void setGameStarted(boolean gameStarted) {
		this.gameStarted = gameStarted;
	}
}
```

Class: CombatPvP.java (Currently Unused)

```
package controller;

import java.util.ArrayList;
```

```java
import java.util.List;
import java.util.Set;

import models.BoardModels.Clearing;
import models.characterModels.PlayerBase;
import models.characterModels.playerEnums.Weights;
import models.otherEntities.EntityBase;
import utils.GameUtils;

public class CombatPvP {
	private List<Clearing> currClearings;
	private boolean lastRound, currRound;
	private PlayerBase[] currOrder;
	//will be the entry point in the end for pvp combat
	public void combatPvP(PlayerBase[] players){
		currClearings = new ArrayList<Clearing>();
		lastRound = false;
		for(int i = 0; i < players.length; ++i)
			currClearings.add(players[i].getCurrentClearing());
		while(true){
			currRound = false;

			preparation(players);

			for(int i = 0; i < players.length; ++i){
				for(int j = 0; j < currClearings.size(); ++j){
					if(!
currClearings.get(j).getEntitiesInClearing().contains(players[i])){
						currClearings.remove(j);
					}
				}
			}
			if((currRound && lastRound) || (currClearings.size() == 0)){
				break;
			}
			lastRound = currRound;
		}
	}

	/*
	 * Encounter
	 * TODO
	 * Albert - look into preparation
	 * preparation
	 * -determine first character for clearing
	 *
	 */

	private void preparation(PlayerBase[] players){
		//have to see if natives will attack players
		//can increase how the natives will act "buy drinks"
		//non-alert native horses


		luring(players);
	}

	/* luring
	 * -first character turn
```

```
	 * --character can lure unhired of any number to own sheet.
	 * --hirelings can lure only one
	 * --can not be relured to other hirelings
	 * --does not change hidden
	 * -repeat till all characters have had turn/no more the lure
	 * -pick unassigned unhired
	 * --all represented charcters roll
	 * --high roll takes
	 * --if character present move to sheet
	 * --else move to unhired sheet
	 * -repeat until all assigned
	 */

	//TODO
	//Albert - think about the return values and parameters for this

	//assumed can only lure unhidden characters/entity
	private void luring(PlayerBase[] players) {
		for(int i = 0; i < players.length; ++i){
			Set<EntityBase> unhidden =
players[i].getCurrentClearing().getUnhiddenEntities();
			if(unhidden.size() != 0){
				int f =
players[i].checkLure(unhidden);//------------------------------view
				if(f > 0){
					players[i].unHide();
				}
			}
		}

//		if (unassigned non-player controlled entities in clearing){ more
relevant for when monsters
//			if (unhidden in clearing){
//				randomAssignments(players);
//			}
//		}

		deployment(players);
	}

	private void randomAssignments(PlayerBase[] players/*, EntityBase[]
unassigned*/){
		//this will take the unassigned and will assign them to attack one of
the players/hirings
		//for that clearing
		//assigned to UNHIDDEN characters in the clearing
	}

	 /*
	 * Deployment & Charging
	 ***can charge or deploy any order
	 * -first player
	 * --if charge hidden = false
	 * --if hireling deploy?
	 * ---if character
	 * ----distribute hireling in red boxes of character's sheet
	 * ---if denizen
	 * ----moves to own sheet hireling on top of attackers
	 * ----if last attacker on another denizen's sheet other
```

```
	 *		denizen becomes first on thrust pile
	 *		attackers controller places attacker either side up
	 *		when it is moved to be to another attacker on new sheet
	 * ---deployed hireling and target hidden = false
	 * --else if more characters to charge/deploy continue
	 * --else Encounter Actions
	 */

	private void deployment(PlayerBase[] players){
		for(int i = 0; i < players.length; ++i){

//			for each see if they want to charge player
//				if so both unhidden
			players[i].setCharge();//-----------------------------------
			if(players[i].chargingPlayer() != null){
				players[i].chargingPlayer().unHide();
				players[i].unHide();
			}
		}

		encounter(players);
	}

	/* Encounter Action
	 * -character may play any number of enchated magic chits (not this
iteration)
	 * -if charged
	 * --check if more character to do actions
	 * -else
	 * --chose 1 (Alert weapon, run, fly away, or cast spell
	 * ---if not then most abandon any number of belongings(active or not)
	 * --check if more characters to do actions
	 * -repeat till no more
	 */

	private void encounter(PlayerBase[] players) {
		for(int i = 0; i < players.length; ++i){
			if(players[i].chargingPlayer() == null){
				continue;
			} else {
				int f = players[i].selection();//---------------------
				if(f != 0)
					continue;
			}
			players[i].selection2();//-------------------
		}
//		rotate through and check if charged another
//			if so then continue
//			else
//				see if alert weapon, run, or fly (can cast spell here)
//				if not
//					may alert any 1 belonging or abandon any number of
belongings

		currOrder = GameUtils.randomOrder(players);

		melee(players);
	}
```

```
	/* TODO
	 * Albert - Have to do the melee steps
	 * Melee Step
	 * -denizen horses turn galloping side up hirelings horses turn over
	 * -Denizens Target!!!*****
	 * mix attention chits of all involved characters. pick first
	 *
	 */

	private void melee(PlayerBase[] players) {
		//rotate through attention chits and select attack targets

//		will have to go through each player to setup their targets and defence/attack

		for(int i = 0; i < players.length; ++i){
			players[i].selectTarget();//-----------------------------
			//should be visible to everyone and go in order
		}

		//secret attack, manuever, and armor for each character

		for(int i = 0; i < players.length; ++i){
			setTactics(players[i]);//------------------------
		}

		//have to send all information to all players

		//roll for repositioning and change of tactics
		for(int i = 0; i < players.length; ++i){
			int f = players[i].rollTacChange();//------------------------
			if (f > 0){
				setTactics(players[i]);//-----------------------------
			}
		}

		//loops through the players and resolves attacks in each
		for(int i = 0; i < currOrder.length; ++i){
			PlayerBase[] clearingOrder = new PlayerBase[currOrder.length];
			int currNum = 0;
			//loops through all the players to see if they have are in the same clearing
			for(int j = 0; j < currOrder.length; j++){
				if(currOrder[i].getCurrentClearing() == currOrder[j].getCurrentClearing()){
					clearingOrder[currNum++] = currOrder[i];
				}
			}
			//sees if the current players clearing is in the list of current clearings
			//will remove if there and run the melee for that clearing else next
			for(int j = 0; j < currClearings.size(); ++j){
				if(currClearings.get(j) == currOrder[i].getCurrentClearing()){
					currClearings.remove(j);
					resolveAttacks(clearingOrder, currNum);
				}
			}
```

```
            }
      }

      private void setTactics(PlayerBase player){
//          TODO
//          player.setAttack();
//          player.setManuever();
//          player.setArmor();
      }

      //will have to have something in regards to the clearing and attack order
      //all the clearing attacks go to resolve attacks and then when the clearning
is
      //done then go to the next clearing to resolve attacks

      /* Resolve Attacks
       * -start with weapon length = 17
       * -Attack Speed = 0
       * -if first combat weapon length > speed
       * -else speed > length
       * -choose attacker
       * --attack undercut?
       * ---Attack Hits
       * --attack intercept maneuver?
       * ---attack Hits
       * --if attacker is character + weapon
       * ---weapon = Alerted
       * --MORE attacks to resolve at this length and speed?
       * ---choose attacker
       * --more attacks to resolve?
       * ---length-1
       * ---speed+1
       * ---next combat round
       * --Character pay for Fatigue and Wounds
       * --Disengagement
       * --Spoils of Combat
       */


      ////TODO make sure this works properly - Albert
      private void resolveAttacks(PlayerBase[] players, int maxPlayers) {
            PlayerBase[] orderAttacks = new PlayerBase[maxPlayers];
            int firstAttack = 0, currAttackNum = 0;
            int firstAttacker = 0;
            PlayerBase attacker, defender;
            // check how the currplayer is attacking
            // who is the attacker and who is defender

            for(int i = 0; i < maxPlayers; ++i){
                  for(int j = 0; j < maxPlayers; ++j){
                        if(players[j].getWeapon().getWeaponLength() > firstAttack){
                              firstAttacker = j;
                              firstAttack =
players[j].getWeapon().getWeaponLength();
                        }
                  }
                  orderAttacks[currAttackNum++] = players[firstAttacker];
                  firstAttack = 0;
            }
```

```
            attacker = orderAttacks[0];
            defender = attacker.attackList();
            orderAttacks = removeFirst(orderAttacks);

            //check all the lengths of weapons in clearning
            //check length then speed if equal
            //then if same for both both attack and hit the other
            //both can die
            while(true){
                  if(!(defender.isLiving())){
                        continue;
                  }

                  if(!(attacker.isLiving())){
                        continue;
                  }

                  int wounds1 = 0, wounds2 = 0;
                  if(defender.getWeapon().getWeaponLength() ==
attacker.getWeapon().getWeaponLength() &&
                     defender.getWeapon().getWeaponSpeed()  ==
attacker.getWeapon().getWeaponSpeed()){
                        wounds1 = checkAttack(attacker, defender);
                        wounds2 = checkAttack(defender, attacker);
                  }

                  attacker.setWounds(wounds1);
                  defender.setWounds(wounds2);

                  attacker = nextAttacker(players);
                  defender = attacker.attackList();
            }
      }

      private PlayerBase[] removeFirst(PlayerBase[] players){
            int i = 0;
            while(true){
                  players[i] = players[++i];
                  if(players[i] == null)
                        break;
            }
            return players;
      }

      private PlayerBase nextAttacker(PlayerBase[] entities){
            PlayerBase nAttacker = null;
            //check for more attacks at this speed and length

      //    check for more attacks at highest speed
            //currplayer = findPlayer(players);
            return nAttacker;
      }

      private int checkAttack(PlayerBase attacker, PlayerBase defender){
            int damage = 0;
            if (!(defender.checkIfHit(attacker.getAttackDirection()))){
                  damage = attackHits(attacker, defender);
            } else if (attacker.getWeapon().getWeaponSpeed() ==
```

```
defender.getSpeed()) {
                  damage = attackHits(attacker, defender);
            } else {
                  attacker.getWeapon().setAlerted(true);
            }
            return damage;
      }


      /* will return 0 if unwounded
       * 1 if 1 wound
       * 2 if horse dies
       * 3 if defender dies */
      private int attackHits(PlayerBase attacker, PlayerBase defender) {
            Weights harm;
            int addStars = 0;

            harm = attacker.getWeapon().getHarmLevel();

            if (attacker.getWeapon().getStars() > 0) {
                  if (defender.armorBlocks(attacker.getWeapon().getAttack()))
{//have to see what these conditions are
                        addStars = attacker.getWeapon().getStars();
                  }
            }

            if (attacker.getWeapon().hasMissles()){
                  //do some roll thing here
                  //GameUtils.missleTable(roll);
            }


            //TODO
            if (attacker.getStrength() > defender.getStrength()){
                  harm = harm.next();
            }

            for(int i =0; i < addStars; ++i){
                  if(harm != Weights.TREMENDOUS){
                        harm = harm.next();
                  } else {
                        i = addStars;
                  }
            }

            //out till add horses
            /*if(defender.onHorse()){
                  if(harm > defender.getHorse().getVulnerability()){
                        return 2;
                  }
            } else */
            /*
            if(true){//will have to change this to check if the defender is a
character when denizens are added
                  if(defender.checkArmor(harm)){
                        defender.checkIfDamaged(harm);
                        if (harm == Weights.MEDIUM || harm == Weights.HEAVY ||
                                    harm == Weights.TREMENDOUS){
                              return 1;
```

```
                }
            } else {
                if (defender.getVulnerability().compareTo(harm) >= 0){
                    return 3;
                }
                if (!(harm == Weights.NEGLIGABLE)){
                    return 1;
                }
                attacker.getWeapon().setAlerted(false);
            }
        }*/
        /*
         * else {
            if(defender.getVulnerability().compareTo(harm) >= 0){
                return 3;
            }
        }  add this when the denizens are added
        */

        return 0;
}

/* Attack Hits
 * -Determine Harm Level(Harm Letter)
 * -Stars?
 * --attack intercepts armor?
 * ---stars-1
 * --level + stars
 * -Missle Weapon?
 * --missle table roll + level
 * --else fight strength higher?
 * ---level+1
 * -Inflict Harm
 * -target on horse?
 * --harm hits horse
 * --HARMVVUL--
 * ---harm > vul
 * ----target killed
 * ----MORE
 * ---target unharmed
 * ---unalerted Weapon
 * ---MORE
 * -character or denizen?
 * --denizen -> HARMVVUL
 * -hit armor?
 * --armor damaged
 * --if harm >= M
 * ---1 wound
 * --unharmed
 * --MORE
 * -Harm > VUL?
 * --target killed
 * --else harm negligible?
 * ---unharmed
 * ---MORE
 * ---else 1 wound
 * ---MORE
 *
 *
```

```
	*/
}
```

Class: CombatPvPHandler.java

```
package controller;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;

import javax.swing.JOptionPane;

import models.characterModels.PlayerBase;
import models.characterModels.playerEnums.Attacks;
import models.characterModels.playerEnums.Weights;
import models.chitModels.ArmorChit;
import models.chitModels.ArmorType;
import models.otherEntities.CombatDataContainer;
import utils.Pair;
import views.CombatView;

/**
 * Revised Combat System To Implement In Into The Game
 */
public class CombatPVPHandler {

	private Attacks currentAttack;
	private PlayerBase currentAttacker;
	private PlayerBase currentDefender;
	private ArrayList<PlayerBase> combattingPlayers;

	private Set<Pair<PlayerBase, PlayerBase>> readyPlayers;
	private Set<CombatDataContainer> combatData;

	private CombatView parentView;

	// Constructor For The Combat Handler
	public CombatPVPHandler (ArrayList <PlayerBase> combattingPlayers, CombatView
parentView) {
		this.combattingPlayers = combattingPlayers;
		currentAttacker = getFirstAttacker();

		currentAttack = null;
		combatData = new HashSet<CombatDataContainer> ();
		readyPlayers = new HashSet<>();

		this.parentView = parentView;
	}

	// Gets The Next Attacker
	public void setNextAttacker () {
		combatData.add(new CombatDataContainer(currentAttacker, currentAttack,
null));
		readyPlayers.add(new Pair<PlayerBase, PlayerBase>(currentAttacker,
currentDefender));

		// If All The Players Have Submitted Then Start The Attack
		if (readyPlayers.size() == combattingPlayers.size()) {
			executeAttacks();
```

```
            }

            int currentPlayerIndex = combattingPlayers.indexOf(currentAttacker);
            currentAttacker = combattingPlayers.get((currentPlayerIndex + 1) %
combattingPlayers.size());
      }

      // Executes The Combat When Everyone Is Done
      public void executeAttacks () {

            parentView.println("");
            parentView.println("Beginning Combat:");

            // Execute The Attacks For The Players
            for (Pair<PlayerBase, PlayerBase> p: readyPlayers) {
                  PlayerBase attacker = p.getFirst();
                  PlayerBase defender = p.getSecond();
                  CombatDataContainer attackerData = findContainer(attacker);
                  CombatDataContainer defenderData = findContainer(defender);

                  // Gather The Armor That Might Have Been Hit On The Defender
                  ArmorChit armorHit =
checkIfArmorProtects(defenderData.getThePlayer(), attackerData.getAttack());

                  // If The Player's Armor Doesn't Prevent Damage Then They Get Hit
                  if (armorHit == null){
                        boolean defenderDead = checkForPlayerDeath (attackerData,
defender);

                        // If The Defender Died Let The Player Now
                        if (defenderDead) {
                              parentView.println("   --- Player " +
defender.getName() + " Has Died");
                              JOptionPane.showMessageDialog(parentView, "Player " +
defender.getName() + " Has Died");
                              combattingPlayers.remove(defender);

                              // If We have Only One Player Left Then Close This
                              if (combattingPlayers.size() < 2) {
                                    parentView.dispose();
                                    break;
                              }
                        }
                  } else {
                        Weights weaponWeight = calculateWeaponHarm(attackerData,
defender);
                        hitPlayersArmor(defender, armorHit, weaponWeight.prev());

                        parentView.println("   --- Hit " + defender.getName() + "'s
" + armorHit.getArmourType());
                        parentView.println("   --- Armour Status: " +
armorHit.getArmorStatus());
                  }
            }

            // Now Purge The List Of Ready Players And Get Ready For The Next Round
            readyPlayers.clear();
      }
```

```
	// See If Player Dies, Do The According Stuff
	private boolean checkForPlayerDeath (CombatDataContainer attackingData,
PlayerBase defendingPlayer){
		Weights weaponWeight = calculateWeaponHarm (attackingData,
defendingPlayer);

		// If Harm Is Higher Then Or Equal To Vulernability Then The Player Is
Dead
		if (weaponWeight.getWeightValue() >=
defendingPlayer.getVulnerability().getWeightValue()) {
			attackingData.getThePlayer().killPlayer(defendingPlayer);
			return true;
		} else {
			return false;
		}
	}

	// Hit This Armor So Register The Hit
	private void hitPlayersArmor (PlayerBase p, ArmorChit a, Weights
incomingDamage) {
		// Register The Damage In The Correct Locations
		if (incomingDamage.getWeightValue() >
a.getArmourWeight().getWeightValue()) {
			a.destoryArmor();
		}
		else if (incomingDamage.getWeightValue() ==
a.getArmourWeight().getWeightValue()) {
			a.damageArmor();
		}

		// If The Armor Is Destoryed Then Remove It
		if (a.isDestoryed()) {
			p.getArmorChits().remove(a);
		}
	}

	// Gathers The First Player For The Combat
	private PlayerBase getFirstAttacker () {
		PlayerBase returnVal = combattingPlayers.get(0);
		return returnVal;
	}

	// Alerts The Current Player Weapon
	public void alertPlayerWeapon () {
		currentAttacker.getWeapon().setAlerted(true);
	}

	// Calculates The Weapon's Harm Value
	private Weights calculateWeaponHarm(CombatDataContainer attacker, PlayerBase
defender) {
		int addStars = 0;
		PlayerBase theAttacker = attacker.getThePlayer();
		Weights harm = theAttacker.getWeapon().getWeaponDamage();

		// Find The Stars Amount For The Player's Attack
		if (theAttacker.getWeapon().getSharpnessStars() > 0) {
			addStars = theAttacker.getWeapon().getSharpnessStars();

			// If We Hit Armor, Then Sharpness Goes Down One Star
```

```
                  if (checkIfArmorProtects(defender, attacker.getAttack()) != null)
{
                        addStars = Math.max(0,
theAttacker.getWeapon().getSharpnessStars());
                  }
            }

            // If The Attacker's Strength Is Higher, Increment Harm level
            if (theAttacker.getStrength() > defender.getStrength()){
                  harm = harm.next();
            }

            // Next Calculate The Harm Level Based On The Stars
            for(int i =0; i < addStars; ++i){
                  harm = harm.next();
            }
            return harm;
      }

      // Check To See If The Armor Protects
      private ArmorChit checkIfArmorProtects (PlayerBase defendingPlayer, Attacks
incomingAttack) {
            ArrayList<ArmorChit> playerArmor = defendingPlayer.getArmorChits();

            // Iterate Over The Player's Armor And See If Blocks
            for (ArmorChit a: playerArmor) {
                  if (a.getArmourType() == ArmorType.BREASTPLATE && (incomingAttack
== Attacks.THRUST || incomingAttack == Attacks.SWING)) {
                        return a;
                  }
                  else if (a.getArmourType() == ArmorType.HELMET && incomingAttack
== Attacks.SMASH) {
                        return a;
                  } else if (a.getArmourType() == ArmorType.ARMOUR) {
                        return a;
                  }
            }

            return null;
      }

      // Find The Data Container Assoicated With The Player
      private CombatDataContainer findContainer (PlayerBase p) {
            for (CombatDataContainer c: combatData) {
                  if (c.getThePlayer() == p) {
                        return c;
                  }
            }
            return null;
      }

      /*---------------------------- Getters And Setters ---------------------- */
      public void setCurrentAttack (Attacks attack) {
            currentAttack = attack;
      }

      public Attacks getCurrentAttack () {
            return currentAttack;
      }
```

```
	public PlayerBase getCurrentAttacker() {
		return currentAttacker;
	}

	public int getReadyPlayerNum () {
		return readyPlayers.size();
	}

	public void setCurrentAttacker(PlayerBase currentAttacker) {
		this.currentAttacker = currentAttacker;
	}

	public PlayerBase getCurrentDefender() {
		return currentDefender;
	}

	public void setCurrentDefender(PlayerBase currentDefender) {
		this.currentDefender = currentDefender;
	}
}
```

Class: DayController.java

```
package controller;

import models.BoardModels.Clearing;
import models.characterModels.PlayerBase;

public class DayController {
	//for the actual controls of a game that loops for the days

	/*
	 * information on this is from pg 33 - 53 then combat
	 *
	 * going to need an unhidden array of players and natives
	 *
	 * end of the day the tile that the character is on will
	 * reveal chits from their pile
	 *
	 * there is a monster roll for each day to determine prowling monsters
	 * may arrive at end of turns/in the evening for battle with characters
	 *
	 * if there are prowling monsters already on board will move to character's
clearing to block
	 * (unhidden only) players
	 * players can run/avoid/fight monsters
	 *
	 * dwellings in valley or woods tiles can contain natives for trade with
	 * players can also hire natives to fight/search for things
	 *
	 * unfriendly and enemy natives mayblock and battle a character unexpectedly
	 *
	 * at some treasure sites/native groups visitors CAN be found
	 * these visitors can trade gold/spells/items gold/missions/campaigns(can
have native groups as allies)
	 * visitors can not be hired/battled
	 *
	 * people can work together and can backstab
	 *
```

```
	 * monsters and natives can be hired and are considered denizen
	 *
	 * some native groups (garrison natives) are the order/Guard/Soldiers/Rogues
	 *
	 * native horses can not be used by anyone other then the native
	 */

	private int day;
	private int maxDays;
	private int maxPhases = 4;
	private PlayerBase[] allPlayers;
	private PlayerBase[] orderPlayers;
	private int totalPlayers;
	private Clearing activeClearings;
	//some view here currentView;

	public void startGame(PlayerBase[] newPlayers){
		day = 0;
		maxDays = 28;
		allPlayers = newPlayers;
		//will have to change this to have hired leaders
		totalPlayers = newPlayers.length;

		while(day != maxDays){
			++day;
			birdSong();
			dayLight();
			evening();
			midnight();
		}

		// check all victory things
	}

	private void birdSong(){
		for(int i = 0; i < totalPlayers; ++i){
			// allPlayers[i].record();

			//have to pick 2-4 phases out of 10
			/* the 10 are
			 * -move (adjacent clearing must be named)
			 * -hide (roll required)
			 * -alert
			 * -rest (1 star of action chit can change for wounded to
inactive or inactive to active
			 *       can do with more stars but must pay back)
			 *       (if unable to rest and no active chits then dead/if all
wounded then dead)
			 * -search (have to search to find the location or the secret
path)
			 *         (you can also look abandoned belongings or locations
discovered)
			 * -trade (can be cancelled)
			 * --buying (have to roll to see if can buy treasure, this is
based
			 *        on how friendly that player is to the natives and can
improve this
			 *        by buying a round of drinks to help one roll, this
costs 1 gold per
```

```
                  *        native on the clearing)(if allowed to buy then have to
trade and/or
                  *        pay the gold required, can get for free, if not will
can decline
                  *        MUST try to buy)
                  *        (fame is also a part of this and will go up if sold and
down if bought)
                  * -hire
                  * -follow
                  * -fly
                  * -use spell
                  *
                  */
                  allPlayers[i].unHide();
            }
            //will have to add things about moving all hired to
            //same place

            //add a thing for prowling creatures
            if(day%7 == 0){
                  //natives and monsters respawn
            }
      }

      private void dayLight(){
            /*
            //this orders the players for the day
            //will have to change this to add for swordsman wanting to go
            for(int i = 0; i < totalPlayers; ++i){
                  int turn = GameUtils.createRandomInt(0, allPlayers.length);
                  orderPlayers[orderPlayers.length] = allPlayers[turn];
                  for(int f = turn; turn < allPlayers.length; ++f){
                        allPlayers[f] = allPlayers[f + 1];
                  }
            }
            allPlayers = orderPlayers;

            for(int i = 0; i < totalPlayers; ++i){
                  orderPlayers[i].unHide();
                  for(int p = 0; p < orderPlayers[i].getMaxPhases(); ++p){//have to
add phases and max phases with the record
                        //also will have to check if illegal moves
//                      trade with players in clearing
//                      rearange belongings
                        switch (orderPlayers[i].getPhase(p)){
                        case "H":
//                            roll on hide table
                            break;
                        case "M":
//                            move to the clearing specified
                            break;
                        case "S":
//                            can search on search table therefore roll
//                            can loot things if discovered or if abandoned items
roll loot table
//                            loot table is a usual roll and then count down from
the top of the stack
//                            if roll > stack = nothing
//                            GameUtils.seachTable(selected, roll)
```

```
                              break;
                        case "T":
//                            trading with certain people gets initiated
                              break;
                        case "R":
//                            have to do something with the rest can do later
                              break;
                        case "A":
                              orderPlayers[i].getWeapon().setAlerted(true);
                              break;
                        case "F":
                              //will set the following to true and the player they
wish to follow
                              break;

                        }

//                      blocking might occur
                  }

//                for(int p = 0; p < totalPlayers; ++p){
//                      check if want to block
//                }

//                map chits in tile to clearing where needed
//                more to do with monsters and natives when that needs to be added
            }

//          warning and sound chit become active
//          more things about monsters moving to hexes
 */
      }

      private void evening(){
            CombatPvP combat = new CombatPvP();
            combat.combatPvP(orderPlayers);
      }

      private void midnight(){
            for(int i = 0; i < totalPlayers; ++i){
//                orderPlayers[i].rearrangeBelongings();
                  orderPlayers[i].getWeapon().setAlerted(false);
            }
//          curses removed from players around chapel tile
      }
}
```

Package: Managers

Class: CharacterTurnManager.java (Currently Not Used)

```
package managers;

import models.characterModels.PlayerBase;

/**
 * Main class that will handle the player's turn when implemented
 * Basically serves as a layer that will determine player action based on the input
given
 * @author Mitchell
```

```
 */
public class CharacterTurnManager {

	private String commandString;
	private final String commandDelmiter = ",";
	private PlayerBase currentCharacter;

	// Base Constructor For The Manager
	public CharacterTurnManager (PlayerBase p, String commandString) {
		currentCharacter = p;
	}
}
```

Package: Models/BoardModels

Class: Clearing.java

```
package models.BoardModels;

import java.awt.Image;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;
import java.util.TreeSet;

import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JButton;

import utils.GameUtils;
import models.characterModels.PlayerBase;
import models.otherEntities.EntityBase;
import models.otherEntities.TreasureModel;

/*
 * Object that represents the logic behind the clearing objects
 * Used mainly for the character's movement but can be expanded on if needed
 */
public class Clearing {

	private boolean blocked;
	private String clearingName;
	private JButton buttonTiedToClearing;
	private Set <Clearing> connectedClearings;
	private Set <TreasureModel> treasuresInClearing;
	private Set <EntityBase> entitiesInClearing;
	private Set <PlayerBase> playersInClearing;
	private Set <Image> imageEnitiesOnThis;

	public Clearing (String clearingName) {
```

```
        blocked = false;
        this.clearingName = clearingName;

        entitiesInClearing = new HashSet<>();
        connectedClearings = new HashSet<>();
        treasuresInClearing = new HashSet<>();
        playersInClearing = new HashSet<>();
        imageEnitiesOnThis = new HashSet<>();

        // Create The Button Tied To The Clearing
        buttonTiedToClearing = new JButton("");
        buttonTiedToClearing.setSize(30, 30);
        buttonTiedToClearing.setOpaque(false);
        buttonTiedToClearing.setContentAreaFilled(false);

        // Create Some Random Treasures For This
        genRandomTreasures();
}

// Player Movement To The Clearing
public void playerMovedToThis (PlayerBase p) {
        blocked = true;

        // Move Off Of The Current Clearing And Onto The Next
        p.setCurrentClearing(this);

        // Set The Button's Icon The Player's Class Image
        addImageToList(p.getImage());

        // Add Them To The Entity Listing
        entitiesInClearing.add(p);
        playersInClearing.add(p);

        updateImage();
}

// Player Moved Away From Clearing
public void playerMovedOffOf (PlayerBase p) {
        entitiesInClearing.remove(p);
        playersInClearing.remove(p);
        removeImageToList(p.getImage());
        updateImage();
}


// Checks Player Movement Against The Adjancent Tiles
public boolean isVaildMove (Clearing c) {
        return connectedClearings.contains(c);
}
```

```
	/// Highlight The Clearing If There Isn't An Image For It
	public void highlightForMove () {
		// If There Is Not An Image On This Tile
		if (imageEnitiesOnThis.size() == 0) {
			JButton clearingButton = getButtonTiedToClearing();
			try {
				Image highlight =
ImageIO.read(getClass().getResource("/moveableClearing.png"));
				Image icon =
highlight.getScaledInstance(buttonTiedToClearing.getWidth(), buttonTiedToClearing.getHeight(),
Image.SCALE_SMOOTH);
				clearingButton.setIcon(new ImageIcon(icon));
				clearingButton.repaint();
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	}

	// Highlight The Connected Clearings If They Don't Contain An Image
	public void highlightConnectedClearings () {
		for (Clearing c: connectedClearings) {
			c.highlightForMove();
		}
	}

	// Add The Treasures To This Clearing
	public void addTreasures (TreasureModel ... treasures) {
		for (TreasureModel t: treasures) {
			treasuresInClearing.add(t);
		}
	}

	// Generate Some Random Treasure Because We Can
	private void genRandomTreasures () {
		// Wheither Or Not We Should Have Treasures On This Clearing (20% chance)
		if (Math.random() <= 1) {
			int numOfTreasures = GameUtils.createRandomInt(1, 5);

			for (int i = 0; i < numOfTreasures; i++) {
				boolean greatTreasure = (Math.random() < 0.2);
				addTreasures(new TreasureModel(greatTreasure));
			}
		}
	}

	// Player Search's Clearing When Called
	public ArrayList<TreasureModel> searchClearing(PlayerBase p) {
```

```
		int dieRoll = GameUtils.createRandomInt(1, 6);
		ArrayList <TreasureModel> returnVal = new ArrayList<>();

		// Temp Holder For Treasure
		if (dieRoll < 4) {
			for (TreasureModel t : treasuresInClearing) {
				t.playerFound(p);
				returnVal.add(t);
			}
		}

		return returnVal;
	}

	// Loots The Clearing And Grabs All Treasure Out
	public void playerLootClearing (PlayerBase p) {
		ArrayList<TreasureModel> treasuresFound = getTreasuresPlayerFound(p);
		for (TreasureModel t: treasuresFound) {
			treasuresInClearing.remove(t);
			p.addTreasure(t);
		}
	}

	// Get The Treasure In The Clearing The PLayer Knows About
	public ArrayList<TreasureModel> getTreasuresPlayerFound(PlayerBase p) {
		ArrayList<TreasureModel> treasures = new ArrayList<>();
		for (TreasureModel t: treasuresInClearing) {
			if (t.hasPlayerFound(p))
				treasures.add(t);
		}
		return treasures;
	}

	// When Adding An Image
	public void addImageToList (Image i) {
		imageEnitiesOnThis.add(i);
		updateImage();
	}

	// When Remove An Image Then Update As Well
	public void removeImageToList (Image i) {
		imageEnitiesOnThis.remove(i);
		updateImage();
	}

	// Reupdates The Tiles Image With What Is In The List
	public void updateImage () {
		ImageIcon displayIcon = new ImageIcon();
```

```
                int width = buttonTiedToClearing.getWidth();
                int height = buttonTiedToClearing.getHeight();

                // If There Only One Image
                if (imageEnitiesOnThis.size() == 1) {
                        Image display = imageEnitiesOnThis.iterator().next();
                        Image icon = display.getScaledInstance(width, height,
Image.SCALE_SMOOTH);
                        displayIcon = new ImageIcon(icon);
                } else if (imageEnitiesOnThis.size() > 1){
                        try {
                                Image icon =
ImageIO.read(getClass().getResource("/custom/characters/question.gif"));
                                Image multiDisplayIcon = icon.getScaledInstance(width, height,
Image.SCALE_SMOOTH);
                                displayIcon = new ImageIcon(multiDisplayIcon);
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                }

                buttonTiedToClearing.setIcon(displayIcon);
                buttonTiedToClearing.repaint();
        }

        // Update All Of The Connected Tiles
        public void updateConnectedTiles () {
                for (Clearing c: connectedClearings) {
                        c.updateImage();
                }
        }

        /* -------------------- Getters And Setters Below Here --------------------*/
        public boolean isBlocked() {
                return blocked;
        }

        public Set <EntityBase> getEntitiesInClearing () {
                return entitiesInClearing;
        }

        public JButton getButtonTiedToClearing() {
                return buttonTiedToClearing;
        }

        public Set <EntityBase> getUnhiddenEntities () {
                TreeSet<EntityBase> returnVal = new TreeSet<>();
                for (EntityBase e: entitiesInClearing) {
                        if (!e.isHidden())
```

```
                        returnVal.add(e);
        }
        return returnVal;
}

public void setBlocked(boolean blocked) {
        this.blocked = blocked;
}

public void setButtonTiedToClearing(JButton buttonTiedToClearing) {
        this.buttonTiedToClearing = buttonTiedToClearing;
}

public void setLocation (int x, int y) {
        buttonTiedToClearing.setLocation(x, y);
}

public Set<Clearing> getConnectedClearings() {
        return connectedClearings;
}

public void setConnectedClearings(Set<Clearing> connectedClearings) {
        this.connectedClearings = connectedClearings;
}

public void addConnectedClearing (Clearing newClearing) {
        this.connectedClearings.add(newClearing);
        newClearing.getConnectedClearings().add(this);
}

// Multi Clearing Connection
public void addToConnectedClearings (Clearing ... clearings) {
        for (Clearing c: clearings) {
                this.addConnectedClearing(c);
        }
}

public String getClearingName() {
        return clearingName;
}

public void setClearingName(String clearingName) {
        this.clearingName = clearingName;
}

public ArrayList<Image> getImageEnitiesOnThis() {
        ArrayList<Image> images = new ArrayList<>();
        for (Image i: imageEnitiesOnThis) {
                images.add(i);
```

```
            }

            return images;
        }

        public void setImageEnitiesOnThis(Set<Image> imageEnitiesOnThis) {
            this.imageEnitiesOnThis = imageEnitiesOnThis;
        }

        public ArrayList<PlayerBase> getPlayersInClearing() {
            ArrayList<PlayerBase> returnVal = new ArrayList<>();
            for (PlayerBase p: playersInClearing) {
                returnVal.add(p);
            }
            return returnVal;
        }

        public void setPlayersInClearing(HashSet<PlayerBase> playersInClearing) {
            this.playersInClearing = playersInClearing;
        }
}
```

Class: Dwelling.java

```
package models.BoardModels;

import java.awt.Image;

/**
 * The Dwelling Objects That A Clearing Can Have On It
 * @author Mitchell
 */
public class Dwelling {
      private Clearing clearingThisOn;
      private Image imageRepresentation;

      // Constructor For The Clearing Object
      public Dwelling (Clearing theClearing, Image imageRep) {
            clearingThisOn = theClearing;
            imageRepresentation = imageRep;
      }

      /*------------------ Getters And Setters ----------------*/
      public Clearing getClearingThisOn() {
            return clearingThisOn;
      }

      public void setClearingThisOn(Clearing clearingThisOn) {
            this.clearingThisOn = clearingThisOn;
      }

      public Image getImageRepresentation() {
            return imageRepresentation;
      }
```

```
	public void setImageRepresentation(Image imageRepresentation) {
		this.imageRepresentation = imageRepresentation;
	}


}
```

Package: Models/CharacterModels/PlayerEnums

Class: Attacks.java

```
package models.characterModels.playerEnums;

public enum Attacks {
	THRUST,
	SWING,
	SMASH
}
```

Class: CharacterClass.java

```
package models.characterModels.playerEnums;

import java.awt.Image;
import java.io.File;
import java.util.ArrayList;

import javax.imageio.ImageIO;

import models.chitModels.ArmorChit;
import models.chitModels.ChitFactory;
import models.chitModels.WeaponChit;

/**
 * A Basic Enumeration That Will Allow For Character Specific
 * Functionality If Needed
 * @author Mitchell
 */
public enum CharacterClass {
   AMAZON,
   BLACKNIGHT,
   CAPTAIN,
   DWARF,
   ELF,
   SWORDSMAN;

   // Method That Will Return The Ready Tile For The Class
   public Image getReadyTile () {
         String baseDir = System.getProperty("user.dir");
            try {
                  switch (this) {
                  case AMAZON:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/amazonUnhidden.png"));
                  case BLACKNIGHT:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/black_knightUnhidden.png")
);
                  case CAPTAIN:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/captainUnhidden.png"));
```

```
                case DWARF:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/dwarfUnhidden.png"));
                case ELF:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/elfUnhidden.png"));
                case SWORDSMAN:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/swordsmanUnhidden.png"));
                default:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/amazonUnhidden.png"));
                }
        } catch (Exception e) {
            e.printStackTrace();
        }
           return null;
   }

   // Method That Will Return The Ready Tile For The Class
   public Image getHiddenTile () {
         String baseDir = System.getProperty("user.dir");
           try {
                switch (this) {
                case AMAZON:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/amazonHidden.png"));
                case BLACKNIGHT:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/black_knightHidden.png"));
                case CAPTAIN:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/captainHidden.png"));
                case DWARF:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/dwarfHidden.png"));
                case ELF:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/elfHidden.png"));
                case SWORDSMAN:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/swordsmanHidden.png"));
                default:
                        return
ImageIO.read(getClass().getResource("/characterGameTiles/amazonHidden.png"));
                }
        } catch (Exception e) {
            e.printStackTrace();
        }
           return null;
   }

   public Image getDetailImage () {
         String baseDir = System.getProperty("user.dir");
           try {
                switch (this) {
                case AMAZON:
                        return
ImageIO.read(getClass().getResource("/characterdetail/amazon.jpg"));
```

```
                case BLACKNIGHT:
                        return
ImageIO.read(getClass().getResource("/characterdetail/black_knight.jpg"));
                case CAPTAIN:
                        return
ImageIO.read(getClass().getResource("/characterdetail/captain.jpg"));
                case DWARF:
                        return
ImageIO.read(getClass().getResource("/characterdetail/dwarf.jpg"));
                case ELF:
                        return
ImageIO.read(getClass().getResource("/characterdetail/elf.jpg"));
                case SWORDSMAN:
                        return
ImageIO.read(getClass().getResource("/characterdetail/swordsman.jpg"));
                default:
                        return
ImageIO.read(getClass().getResource("/characterdetail/amazon.jpg"));
                }
        } catch (Exception e) {
              e.printStackTrace();
        }
           return null;
   }

      public WeaponChit getStartingWeapon() {
            switch (this) {
            case AMAZON:
                  return ChitFactory.LIGHT_BOW;
            case BLACKNIGHT:
                  return ChitFactory.SPEAR;
            case CAPTAIN:
                  return ChitFactory.SPEAR;
            case DWARF:
                  return ChitFactory.AXE;
            case ELF:
                  return ChitFactory.LIGHT_BOW;
            case SWORDSMAN:
                  return ChitFactory.SPEAR;
            default:
                  return ChitFactory.SPEAR;
            }
   }

      public Weights getVulner () {
            switch (this) {
            case AMAZON:
                  return Weights.MEDIUM;
            case BLACKNIGHT:
                  return Weights.MEDIUM;
            case CAPTAIN:
                  return Weights.MEDIUM;
            case DWARF:
                  return Weights.HEAVY;
            case ELF:
                  return Weights.LIGHT;
            case SWORDSMAN:
                  return Weights.LIGHT;
            default:
```

```
                return Weights.LIGHT;
            }
        }

        // Gather The Armour Chits For This Player Class
        public ArrayList<ArmorChit> getArmour () {
            ArrayList<ArmorChit> returnVal = new ArrayList<>();
            switch (this) {
            case AMAZON:
                returnVal.add(ChitFactory.getHelmet());
                returnVal.add(ChitFactory.getShield());
                returnVal.add(ChitFactory.getBreastPlate());
                break;
            case BLACKNIGHT:
                returnVal.add(ChitFactory.getHelmet());
                returnVal.add(ChitFactory.getBreastPlate());
                break;
            case CAPTAIN:
                returnVal.add(ChitFactory.getShield());
                break;
            case DWARF:
                returnVal.add(ChitFactory.getHelmet());
                break;
            case ELF:
                break;
            case SWORDSMAN:
                break;
            default:
                break;
            }
            return returnVal;
        }
}
```

Class: TradeRelation.java

```
package models.characterModels.playerEnums;

public enum TradeRelation {
        ENEMY,
        UNFRIENDLY,
        NEUTRAL,
        FRIENDLY,
        ALLY
}
```

Class: Weights.java

```
package models.characterModels.playerEnums;

//specific weights for character classes

public enum Weights {
        NEGLIGABLE(0),
        LIGHT(1),
        MEDIUM(2),
        HEAVY(3),
        TREMENDOUS(4);

        private int WEIGHT_VALUE;
```

```
	Weights (int weightValue) {
		WEIGHT_VALUE = weightValue;
	}

	public int getWeightValue (){
		return WEIGHT_VALUE;
	}

	public Weights next() {
		if (this == TREMENDOUS) {
			return TREMENDOUS;
		}
		return values()[(this.ordinal() + 1) % values().length];
	}

	public Weights prev() {
		if (this == NEGLIGABLE) {
			return NEGLIGABLE;
		}
		return values()[(this.ordinal() - 1) % values().length];
	}

}
```

Package: Models/CharacterModels/PlayerEnums

Class: Amazon.java

```
package models.characterModels;

import models.characterModels.playerEnums.CharacterClass;
import models.characterModels.playerEnums.Weights;

public class Amazon extends PlayerBase {

	public Amazon(){
		vulnerability = Weights.MEDIUM;
		setClass(CharacterClass.AMAZON);
	}
}
```

Class: BlackNight.java

```
package models.characterModels;

import models.characterModels.playerEnums.CharacterClass;
import models.characterModels.playerEnums.Weights;
import models.chitModels.Chit;

public class BlackKnight extends PlayerBase {
	public BlackKnight(Chit[] combat){
		super();
		vulnerability = Weights.MEDIUM;
		setClass(CharacterClass.BLACKNIGHT);
	}
}
```

Class: Captain.java

```
package models.characterModels;
```

```
import models.characterModels.playerEnums.CharacterClass;
import models.characterModels.playerEnums.Weights;

public class Captain extends PlayerBase {
	public Captain() {
		super();
		vulnerability = Weights.MEDIUM;
		setClass(CharacterClass.CAPTAIN);
	}
}
```

Class: Dwarf.java

```
package models.characterModels;

import models.characterModels.playerEnums.CharacterClass;
import models.characterModels.playerEnums.Weights;

public class Dwarf extends PlayerBase {
	public Dwarf() {
		super();
		vulnerability = Weights.HEAVY;
		setClass(CharacterClass.DWARF);
	}
}
```

Class: Elf.java

```
package models.characterModels;

import models.characterModels.playerEnums.CharacterClass;
import models.characterModels.playerEnums.Weights;

public class Elf extends PlayerBase {
	public Elf(){
		super();
		vulnerability = Weights.LIGHT;
		setClass(CharacterClass.ELF);
	}
}
```

Class: PlayerBase.java

```
package models.characterModels;

import java.awt.Image;
import java.util.ArrayList;
import java.util.HashSet;
import java.util.Set;

import javax.swing.ImageIcon;
import javax.swing.JButton;

import models.BoardModels.Clearing;
import models.characterModels.playerEnums.Attacks;
import models.characterModels.playerEnums.CharacterClass;
import models.characterModels.playerEnums.Weights;
import models.chitModels.ActionChit;
import models.chitModels.ArmorChit;
import models.chitModels.Chit;
```

```java
import models.chitModels.WeaponChit;
import models.otherEntities.Denizen;
import models.otherEntities.EntityBase;
import models.otherEntities.TreasureModel;
import utils.GameUtils;

/*
 * will have the initial information about the player
 * such as their character
 * history
 * inventory etc.
 * testing
 */

public class PlayerBase extends EntityBase{

	//fame/not can be negative and are public to others, gold can't be neg
	protected int currentFame;
	protected int currentNotirity;
	protected int currentGold;

	//weapons/armor/horses/treasure cards
	//reading runes at treasure sites
	//magical artifacts/spell books
	//fighting and killing monsters, natives, or characters
	//winner of the game is the one with most points above the amount to win
	//i.e. some one that needs 20 and gets 21 will lose to someone that
	//    need 15 and gets 17
	protected int vicotryPoints;
	protected int winVictoryPoints;
	protected int killCount;
	protected int availableActions;

	// Player's Name And Class
	protected String playerName;
	protected String playerClass;
	protected String tradeRelationship;

	//Action Chits in different statuses
	protected ActionChit[] Active;
	protected ActionChit[] InActive;
	protected ActionChit[] Wounded;

	// Compound Data Types For The Object
	protected Chit[] combatChit;
	protected Chit horse;
	protected Weights wounds;
	protected Weights vulnerability;
	protected CharacterClass characterClass;
	protected WeaponChit activeWeapon;

	// Lists For The Player
	protected ArrayList<ArmorChit> armorChits;
	protected ArrayList<WeaponChit> weaponChit;
	protected ArrayList<TreasureModel> accquiredTreasures;
	protected ArrayList<PlayerBase> listAttacks;
	protected PlayerBase chargeTarget;

	// Clearing Stuff
```

```
protected Clearing homeClearing;
protected Clearing currentClearing;

// Log Recording
protected String currentTurn;
protected ArrayList<String> turnLog;

//controllable units might need might not
//big part of combat system
//might also include natives?
protected Denizen[] hired;
protected Image hiddenImage;
protected Image unhiddenImage;

protected String history;
protected String discovery;

// Boolean Flags
protected boolean moving;
protected boolean living;
protected boolean foundHidden;

// Default Constructor
public PlayerBase () {
	initPlayerStats();
}

public PlayerBase (String playerName, CharacterClass c) {
	setName(playerName);
	setClass(c);
	initPlayerStats();
}


//sould have a starting location on a dwelling
// Initialize The Player Stats
protected void initPlayerStats () {
	currentFame = 0;
	currentNotirity = 0;
	currentGold = 10;
	killCount = 0;
	availableActions = 5;
	foundHidden = false;
	hidden = false;
	living = true;
	currentTurn = "";

	// Setup the lists
	turnLog = new ArrayList<>();
	accquiredTreasures = new ArrayList<>();

	// Setup The Image
	hiddenImage = characterClass.getHiddenTile();
	unhiddenImage = characterClass.getReadyTile();
}

// Starts The Player's Turn And Wipes Out There Player's Commands
public void startPlayerTurn () {
	hidden = false;
```

```
            currentTurn = "";
            currentClearing.playerMovedToThis(this);

            // Set The Available Actions To 0
            availableActions = 5;
      }

      // Do All The Things For Ending The Player's Turn
      public void endPlayerTurn () {
            turnLog.add(currentTurn);
            currentTurn = "";
      }

      // Attempts To Hide The Player, Consult Die Table For Reasoning For Result
      public boolean attemptHide () {
            availableActions--;
            int dieRoll = GameUtils.createRandomInt(1, 6);
            hidden = (hidden) ? true : dieRoll < 6;

            logAction ("H");

            // If The Player Hid Then Simply Refresh The Player's Image
            if (hidden) {
                  JButton clearingButton =
currentClearing.getButtonTiedToClearing();
                  Image playerIcon =
getImage().getScaledInstance(clearingButton.getWidth(), clearingButton.getHeight(),
Image.SCALE_SMOOTH);
                  currentClearing.getButtonTiedToClearing().setIcon(new ImageIcon
(playerIcon));
                  clearingButton.repaint();
            }

            return hidden;
      }

      // Attempts To Move The Player To The Designated Clearing
      public boolean moveToClearing (Clearing newClearing) {
            if (currentClearing.isVaildMove(newClearing)) {

                  // Log The Turn On The Player's Log And Subtract An Action
                  availableActions--;
                  logAction("M-" + newClearing.getClearingName());

                  updateConnectedClearings();
                  currentClearing.updateImage();
                  currentClearing.playerMovedOffOf(this);
                  currentClearing = newClearing;
                  currentClearing.playerMovedToThis(this);
                  return true;
            } else{
                  return false;
            }
      }

      // Moves The Player Straight To Home When Called
      public void moveToHome() {
            updateConnectedClearings();
            currentClearing.updateImage();
```

```
			currentClearing.playerMovedOffOf(this);
			currentClearing = homeClearing;
			currentClearing.playerMovedToThis(this);
		}

		// Resets The Players Values When Called
		public void reset() {
			hidden = false;
			moving = false;
			updateConnectedClearings();
		}

		// Kills The Player And Adds All Of The Values To This
		public void killPlayer (PlayerBase oppoent) {
			this.currentFame += oppoent.currentFame;
			this.currentGold += oppoent.currentGold;
			this.currentNotirity += oppoent.currentNotirity;
			killCount++;
			oppoent.moveToHome();
		}

		// Treasure Functions
		public void addTreasure (TreasureModel t) {
			accquiredTreasures.add(t);
			currentGold += t.getTreasureGoldValue();
		}

		public ArrayList<TreasureModel> searchCurrentClearing () {
			logAction("S-" + currentClearing.getClearingName());
			availableActions--;
			return currentClearing.searchClearing(this);
		}

		public void logAction (String logMessage) {
			currentTurn += (currentTurn.equals("")) ? logMessage : "," +
logMessage;
		}

		// Updates All Of The Connected
		public void updateConnectedClearings() {
			currentClearing.updateConnectedTiles();
		}

		//can be changed to something else if weight is not an issue;
		//have to change this so that it will do things to the action chits
		public void increaseWounds(int d){
			for(int i = 0; i < d; ++i){
				wounds = wounds.next();
				if (wounds == vulnerability){
					isDead();
					return;
				}
			}
		}

		public void resetWounds(){
			wounds = Weights.NEGLIGABLE;
		}
```

```
	public void setCharge() {
		// TODO Auto-generated method stub
		Set<EntityBase> chargeAble = getSelectable();
		//have to take the chargeAble to view for selection or none
	}

	public void selectTarget(){
		// TODO
		Set<EntityBase> targetAble = getSelectable();
		//have to send to view to see if selection
	}

	private Set<EntityBase> getSelectable(){
		Set<EntityBase> Selectable = new HashSet<EntityBase>();
		if(foundHidden){//something to do with the search phase of the day
			Selectable.addAll(getCurrentClearing().getEntitiesInClearing());
		} else {
			if (getCurrentClearing().getUnhiddenEntities() != null){

	Selectable.addAll(getCurrentClearing().getUnhiddenEntities());
			}
		}
		return Selectable;
	}

	//*********
	//if a player dies will start over again at the in as the same or different
	//character but will be forfeiring all his
possessions/fame/notoriety/gold/discoveries

	/*-------------- Getters And Setters -------------- */
	public ArrayList<String> getRecordLog () {
		return turnLog;
	}

	public void setHomeClearing(Clearing homeClearing) {
		this.homeClearing = homeClearing;
	}

	public void setClass(CharacterClass newPlayerClass) {
		characterClass = newPlayerClass;
		vulnerability = newPlayerClass.getVulner();

		// Weapon For This Player
		weaponChit = new ArrayList<>();
		weaponChit.add(newPlayerClass.getStartingWeapon());
		activeWeapon = weaponChit.get(0);

		// Armor Chits
		armorChits = new ArrayList<>();
		armorChits.addAll(newPlayerClass.getArmour());
		System.out.println();
	}

	public void setName(String newPlayerName){
		playerName = newPlayerName;
	}

	public void setGold (int amount) {
```

```
        currentGold = Math.max(amount, 0);
    }

    public CharacterClass getPlayerClass(){
        return characterClass;
    }

    public Weights getVulnerability(){
        return vulnerability;
    }

    public void setVulnerability(Weights vul){
        this.vulnerability = vul;
    }

    public boolean isMoving() {
        return moving;
    }

    public void setMoving(boolean moving) {
        this.moving = moving;
    }

    public Clearing getCurrentClearing() {
        return currentClearing;
    }

    public void setCurrentClearing(Clearing currentClearing) {
        this.currentClearing = currentClearing;
    }

    public Image getImage() {
        return (hidden) ? hiddenImage : unhiddenImage;
    }

    public String getName(){
        return playerName;
    }

    public int getCurrentFame() {
        return currentFame;
    }

    public int getCurrentNotirity() {
        return currentNotirity;
    }

    public int getCurrentGold() {
        return currentGold;
    }

    public Clearing getHomeClearing() {
        return homeClearing;
    }

    public void addAttack(PlayerBase toAttack){
        listAttacks.add(toAttack);
    }
```

```
public PlayerBase attackList(){
	return listAttacks.remove(0);
}

public boolean isLiving(){
	return living;
}

public void unHide(){
	hidden = false;
}

public WeaponChit getWeapon() {
	return activeWeapon;
}

public ArrayList<ArmorChit> getArmorChits() {
	return armorChits;
}

public void setArmorChits(ArrayList<ArmorChit> armorChits) {
	this.armorChits = armorChits;
}

public int getStrength() {
	// TODO Auto-generated method stub
	return 0;
}

public void checkIfDamaged(Weights harm) {
	// TODO Auto-generated method stub
}

public void isDead() {
	living = false;
}

public int getKillCount() {
	return killCount;
}

public void setKillCount(int killCount) {
	this.killCount = killCount;
}

public int getAvailableActions() {
	return availableActions;
}

public void setAvailableActions(int availableActions) {
	this.availableActions = availableActions;
}

public boolean armorBlocks(Attacks attack) {
	// TODO Auto-generated method stub
	//should loop through what can be blocked in the attack types
	//if blockable attack then return true
	return true;
}
```

```
	//will return 0 if not able to change
	public int rollTacChange() {
		// TODO Auto-generated method stub
		return 0;
	}

	public int selection() {
		// TODO Auto-generated method stub
		//will ask if player wants to alert weapon, run, or fly (can cast spell
here) or nothing
		//will do what is selected and return 1
		//will return 0 if do nothing
		return 0;
	}

	public void selection2() {
		// TODO Auto-generated method stub
		//must alert something or abandon an amount of belongings
	}

	public void toCharge(Set<EntityBase> chargeAble) {
		// TODO Auto-generated method stub
		//will change charge target to something

	}

	public PlayerBase chargingPlayer() {
		return chargeTarget;
	}

	public void setWounds(int wounds) {
		// TODO Auto-generated method stub
		if(wounds == 1){
			increaseWounds(wounds);
		}
		if(wounds == 2){
			//no horse
		}
		if(wounds == 3){
			isDead();
		}

	}

	public Attacks getAttackDirection() {
		// TODO Auto-generated method stub
		//have to set it up so that you can get your weapon to swing a set
direction
		return null;
	}

	public boolean checkIfHit(Attacks attackDirection) {
		// TODO Auto-generated method stub
		// will check manuever vsv attackDirection and if hit then return true
		return false;
	}

	public int getSpeed() {
```

```
		// TODO Auto-generated method stub
		// this is the manuever speed of the player that round
		return 0;
	}

	public int checkLure(Set<EntityBase> unhidden) {
		// TODO Auto-generated method stub
		// will ask the player if they want to lure any monsters/natives
		return 0;
	}

	@Override
	public String toString() {
		return "(Player: " + getName() + ", Class: " + getPlayerClass() + ")";
	}
}
```

Class: Swordsman.java

```
package models.characterModels;

import models.characterModels.playerEnums.CharacterClass;
import models.characterModels.playerEnums.Weights;

public class Swordsman extends PlayerBase {
	public Swordsman(){
		super();
		vulnerability = Weights.LIGHT;
		setClass(CharacterClass.SWORDSMAN);
	}
}
```

Package: ChitModels

Class: ArmorChit.java

```
package models.chitModels;

import models.characterModels.playerEnums.Weights;

public class ArmorChit {

	private ArmorType armourType;
	private boolean damaged;
	private boolean destoryed;
	private Weights armourWeight;

	public ArmorChit (ArmorType aType, Weights aWeight) {
		armourType = aType;
		armourWeight = aWeight;
		damaged = false;
		destoryed = false;
	}

	public void damageArmor () {
		destoryed = (damaged) ? true : destoryed;
		damaged = true;
	}

	public void destoryArmor () {
		destoryed = true;
```

```
	}

	/*---------------- Getters And Setters -------------------*/
	public ArmorType getArmourType() {
		return armourType;
	}

	public void setArmourType(ArmorType armourType) {
		this.armourType = armourType;
	}

	public Weights getArmourWeight() {
		return armourWeight;
	}

	public void setArmourWeight(Weights armourWeight) {
		this.armourWeight = armourWeight;
	}

	public boolean isDamaged() {
		return damaged;
	}

	public void setDamaged(boolean damaged) {
		this.damaged = damaged;
	}

	public boolean isDestoryed() {
		return destoryed;
	}

	public String getArmorStatus () {
		if (destoryed) {
			return "Destroyed";
		} else if (damaged) {
			return "Damaged";
		}

		return "Undamaged";
	}

	public void setDestoryed(boolean destoryed) {
		this.destoryed = destoryed;
	}

	@Override
	public String toString() {
		return armourType.name();
	}
}
```

Class: ArmorType.java

```
package models.chitModels;

public enum ArmorType {
	ARMOUR,
	HELMET,
	SHIELD,
	BREASTPLATE;
```

```
}
```

Class: ChitFactory.java

```
package models.chitModels;

import models.characterModels.playerEnums.Weights;

public class ChitFactory {

	// Weapon Chits
	public static WeaponChit AXE = new WeaponChit("Axe", 2, 1, Weights.MEDIUM);
	public static WeaponChit SPEAR = new WeaponChit("Spear", 10, 1,
Weights.MEDIUM);
	public static WeaponChit LIGHT_BOW = new WeaponChit("Light Bow", 14, 2,
Weights.LIGHT);
	public static WeaponChit THRUSTING_SWORD = new WeaponChit("Thrusting Sword",
4, 1, Weights.LIGHT);

	/* ---------------------------- Armor Chits ---------------------------- */
	public static ArmorChit getHelmet () {
		return new ArmorChit(ArmorType.HELMET, Weights.MEDIUM);
	}

	public static ArmorChit getBreastPlate () {
		return new ArmorChit(ArmorType.BREASTPLATE, Weights.MEDIUM);
	}

	public static ArmorChit getShield () {
		return new ArmorChit(ArmorType.SHIELD, Weights.MEDIUM);
	}

	public static ArmorChit getArmor () {
		return new ArmorChit(ArmorType.ARMOUR, Weights.MEDIUM);
	}
}
```

Class: WeaponChit.java

```
package models.chitModels;

import models.characterModels.playerEnums.Weights;

/**
 * Base Weapon Chit
 * @author Mitchell
 */
public class WeaponChit extends Chit{

	private int weaponLength;
	private int sharpnessStars;
	private int weaponSpeed;
	private String weaponName;
	private Weights weaponHarm;

	private boolean alerted;
	private boolean missleBased;

	// Constructor For The Weapon Chit
	public WeaponChit (String weaponName, int length, int sharpnessLevel, Weights
aWeapDmg) {
```

```
            weaponLength = length;
            sharpnessStars = sharpnessLevel;
            weaponHarm = aWeapDmg;
            this.weaponName = weaponName;

            alerted = false;
            missleBased = false;
      }

      /*-------------------------- Getters And Setters
-----------------------------*/
      public int getWeaponLength() {
            return weaponLength;
      }

      public void setWeaponLength(int weaponLength) {
            this.weaponLength = weaponLength;
      }

      public boolean hasMissles() {
            return missleBased;
      }

      public int getSharpnessStars() {
            return sharpnessStars;
      }

      public void setSharpnessStars(int sharpnessStars) {
            this.sharpnessStars = sharpnessStars;
      }

      public boolean isAlerted() {
            return alerted;
      }

      public void setAlerted(boolean alerted) {
            this.alerted = alerted;
      }

      public Weights getWeaponDamage () {
            return weaponHarm;
      }

      public int getWeaponSpeed() {
            return weaponSpeed;
      }

      public void setWeaponSpeed(int weaponSpeed) {
            this.weaponSpeed = weaponSpeed;
      }

      public String getWeaponName() {
            return weaponName;
      }

      public void setWeaponName(String weaponName) {
            this.weaponName = weaponName;
      }
}
```

Package: OtherEnities

Class: CombatDataContainer.java

```
package models.otherEntities;

import models.characterModels.PlayerBase;
import models.characterModels.playerEnums.Attacks;

/**
 * Just A Small Container That Holds Helpful Data
 * @author Mitchell
 */
public class CombatDataContainer {
	private PlayerBase thePlayer;
	private Attacks attack;
	private Attacks defense;

	public CombatDataContainer (PlayerBase p, Attacks attack, Attacks defense) {
		thePlayer = p;
		this.attack = attack;
		this.defense = defense;
	}

	/*---------------------- Getters And Setters ---------------------------*/
	public PlayerBase getThePlayer() {
		return thePlayer;
	}

	public void setThePlayer(PlayerBase thePlayer) {
		this.thePlayer = thePlayer;
	}

	public Attacks getAttack() {
		return attack;
	}

	public void setAttack(Attacks attack) {
		this.attack = attack;
	}

	public Attacks getDefense() {
		return defense;
	}

	public void setDefense(Attacks defense) {
		this.defense = defense;
	}
}
```

Class: Denizen.java

```
package models.otherEntities;

import models.characterModels.PlayerBase;

public class Denizen extends EntityBase {
	//if hiredBy is null then the ai will have to controll them
	//else controlled by the player that is hiredBy
	private PlayerBase hiredBy;
```

```
	public Denizen(){
		setHiredBy(null);
	}

	//setters and getters
	public PlayerBase getHiredBy() {
		return hiredBy;
	}

	public void setHiredBy(PlayerBase hiredBy) {
		this.hiredBy = hiredBy;
	}

}
```

Class: EntityBase.java

```
package models.otherEntities;

/*
 * super class for all other entities that exist in the world
 * could be natives, monsters, etc.
 */
public class EntityBase {
	protected boolean hidden;

	public boolean isHidden() {
		return hidden;
	}

	public void setHidden(boolean hidden) {
		this.hidden = hidden;
	}
}
```

Class: TreasureModel.java

```
package models.otherEntities;

import java.util.ArrayList;

import utils.GameUtils;
import models.characterModels.PlayerBase;

/**
 * Basic Model For The Treasures In The Game
 * Players find treasure by looking in a clearing and then looting it
 * @author Mitchell
 */
public class TreasureModel {

	private int treasureGoldValue;
	private boolean greatTreasure;
	private ArrayList<PlayerBase> playersFoundThis;

	// Static Array of Values For The Treasure Amounts
	private int[] goldAmounts = {10, 20, 30, 40, 50};

	public TreasureModel (boolean greatTreasure) {
```

```
            playersFoundThis = new ArrayList<PlayerBase>();

            // Determine The Gold Amount, Rando For Now
            int startRange = (greatTreasure) ? 3 : 0;
            int endRange = (greatTreasure) ? goldAmounts.length - 1: 2;
            treasureGoldValue = goldAmounts[GameUtils.createRandomInt(startRange,
endRange)];
      }

      // Adds The Player To The Found Array
      public void playerFound (PlayerBase p) {
            playersFoundThis.add(p);
      }

      // Returns True If The Player Has Found This Treasure
      public boolean hasPlayerFound (PlayerBase p) {
            return playersFoundThis.contains(p);
      }

      public int getTreasureGoldValue() {
            return treasureGoldValue;
      }

      @Override
      public String toString() {
            String displayString = (greatTreasure) ? "Great Treasure - Value: " :
"Normal Treasure - Value: ";
            return displayString + treasureGoldValue;

      }
}
```

Package: Networking (Currently Not Used)

Class: ClientThread.java

```
package networking;

import java.net.Socket;

/**
 * Client Thread That Will Communicate With The Server And Give The Client Updates
 *    - Will Be Expanded Upon In The Future But For Now It's A Skeleton
 * @author Mitchell
 */
public class ClientThread extends TransmissionThreadBase {

      // Specific Server To Connect Top
      public ClientThread (String serverHost, int serverPort) {
            super(serverHost, serverPort);
      }

      // Socket Constructor For The Thread
      public ClientThread(Socket newSocket) {
            super(newSocket);
      }

}
```

Class: ServerMainThread.java

```
package networking;

import java.net.ServerSocket;
import java.net.Socket;

/**
 * Thread That Contains The Actual Server, This Way Clients Can Host There Own Game
 * @author Mitchell
 */
public class ServerMainThread extends Thread {

	int serverPort;

	// Constructor For The Thread
	public ServerMainThread(int port) {
		serverPort = port;
	}

	@Override
	// Main Method That Will Start Up The Server And Handle Everything
	public void run () {
		System.out.println("Starting Server On Port " + serverPort);
		try (ServerSocket serverSocket = new ServerSocket(serverPort);) {

			// Run Indefinitely Waiting For Incoming Connection
			while (true) {
				Socket newSocket = serverSocket.accept();
				System.out.println("New Client Connected: " +
newSocket.getInetAddress().getHostAddress());
			}

		} catch (Exception e) {
			e.printStackTrace();
		}
	}
}
```

Class: ServerThread.java

```
package networking;

import java.net.Socket;

/**
 * Client Thread That Will Communicate With The Server And Give The Client Updates
 *    - Will Be Expanded Upon In The Future But For Now It's A Skeleton
 * @author Mitchell
 */
public class ServerThread extends TransmissionThreadBase {

	// Specific Server To Connect Top
	public ServerThread (String serverHost, int serverPort) {
		super(serverHost, serverPort);
	}

	// Socket Constructor For The Thread
	public ServerThread(Socket newSocket) {
		super(newSocket);
	}
```

```
}
```

Class: TransmissionThreadBase.java

```
package networking;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

public class TransmissionThreadBase extends Thread {

	private Socket transmissionSocket;
	private ObjectInputStream inStream;
	private ObjectOutputStream outStream;

	// IP And Port Constructor
	public TransmissionThreadBase (String host, int port) {
		try {
			transmissionSocket = new Socket (host, port);
			initStreams();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	// Constructor For The Client Thread
	public TransmissionThreadBase(Socket newSocket) {
		transmissionSocket = newSocket;
		initStreams();
	}

	// Setup The Streams For The Connection
	private void initStreams() {
		try {
			// Setup The Streams
			outStream = new ObjectOutputStream(transmissionSocket.getOutputStream());
			inStream = new ObjectInputStream(transmissionSocket.getInputStream());

			// Display The Data Out From The Client
			String clientIP = transmissionSocket.getInetAddress().getHostAddress();
			System.out.println("Client Connected From: " + clientIP);
		} catch (IOException e) {
			System.out.println("An Error Occured With Client Connection...");
			e.printStackTrace();
		}
	}
```

```
// Method Closes The Socket
public void closeConnection() {
	// Close The Input Stream
	try {
		if (inStream != null) {
			inStream.close();
		}
	} catch (Exception e) {
	}

	// Close The Output Stream
	try {
		if (outStream != null) {
			outStream.close();
		}
	} catch (Exception e) {
	}

	// Close The Client Socket
	try {
		if (transmissionSocket != null) {
			transmissionSocket.close();
		}
	} catch (Exception e) {
	}
}

// Send A Message To The Server If The Connection
public boolean writeMsg(Object o) {
	if (!transmissionSocket.isConnected()) {
		closeConnection();
		return false;
	}

	// Send The Object Over To The Server
	try {
		outStream.writeObject(o);
	} catch (IOException e) {
		e.printStackTrace();
	}
	return true;
}

@Override
public void run() {
	while (true) {
		try {
			inStream.readObject();
		} catch (ClassNotFoundException | IOException e) {
```

```
                        closeConnection();
                        break;
                    }
                }
            }
        }
```

Package: Utils

Class: GameUtils.java

```
package utils;

import java.util.Random;

import models.characterModels.PlayerBase;
import models.characterModels.playerEnums.TradeRelation;

/*
 * Utility Class For The Game That Has Static Methods For Ease Of Acesss
 */
public class GameUtils {

    public static String SEARCH_LOCATE = "LOCATE";
    public static String SEARCH_LOOT = "LOOT";

    // Returns A Random Value Between Ranges
    public static int createRandomInt (int beginRange, int endRange) {
        Random randomizer = new Random();
        return (randomizer.nextInt((endRange - beginRange) + 1) + beginRange);
    }

    //Returns A Random Array of Players From Another Array
    public static PlayerBase[] randomOrder(PlayerBase[] currPlayers){
        int total = currPlayers.length;
        PlayerBase[] rList = {};
        for(int i = 0; i < total; ++i){
            int turn = createRandomInt(0, currPlayers.length);
            rList[rList.length] = currPlayers[turn];
            for(int f = turn; turn < currPlayers.length; ++f){
                currPlayers[f] = currPlayers[f + 1];
            }
        }
        return rList;
    }

    //Tables that are constant and all will return a string for result
    public static int missleTable(int roll){
        int result = 0;
        switch (roll){
        case 1:
```

```
                result = 2;
                break;
            case 2:
                result = 1;
                break;
            case 3:
                result = 0;
                break;
            case 4:
                result = -1;
                break;
            case 5:
                result = -2;
                break;
            case 6:
                result = -3;
                break;
            }
            return result;
        }

//      X# = price*#
//      B# = can free
//      OPP = reRoll one greater
//      No = no deal
//      Trouble = to Unfriendly
//      Fight = block/Attack
//      Challenge = fame-5 or Fight
//      Insult = Notoriety-5 or Fight
        public static String meetingTable(TradeRelation relation, int roll){
            String result = null;
            switch (relation){
            case ENEMY:
                result = meetingEnemy(roll);
                break;
            case UNFRIENDLY:
                result = meetingUnfriendly(roll);
                break;
            case NEUTRAL:
                result = meetingNeutral(roll);
                break;
            case FRIENDLY:
                result = meetingFriendly(roll);
                break;
            case ALLY:
                result = meetingAlly(roll);
                break;
            }
            return result;
```

```
}

public static String meetingEnemy(int roll){
	String result = null;
	switch (roll){
	case 1:
		result = "Insult";
		break;
	case 2:
		result = "Challenge";
		break;
	case 3:
		result = "Fight";
		break;
	case 4:
		result = "Fight";
		break;
	case 5:
		result = "Fight";
		break;
	case 6:
		result = "Fight";
		break;
	}
	return result;
}

public static String meetingUnfriendly(int roll){
	String result = null;
	switch (roll){
	case 1:
		result = "X4";
		break;
	case 2:
		result = "No";
		break;
	case 3:
		result = "No";
		break;
	case 4:
		result = "Insult";
		break;
	case 5:
		result = "Challenge";
		break;
	case 6:
		result = "Fight";
		break;
	}
```

```
		return result;
	}

	public static String meetingNeutral(int roll){
		String result = null;
		switch (roll){
		case 1:
			result = "OPP";
			break;
		case 2:
			result = "X3";
			break;
		case 3:
			result = "X4";
			break;
		case 4:
			result = "No";
			break;
		case 5:
			result = "No";
			break;
		case 6:
			result = "Trouble";
			break;
		}
		return result;
	}

	public static String meetingFriendly(int roll){
		String result = null;
		switch (roll){
		case 1:
			result = "OPP";
			break;
		case 2:
			result = "X2";
			break;
		case 3:
			result = "X2";
			break;
		case 4:
			result = "X3";
			break;
		case 5:
			result = "X4";
			break;
		case 6:
			result = "No";
			break;
```

```
        }
        return result;
}

public static String meetingAlly(int roll){
        String result = null;
        switch (roll){
        case 1:
                result = "B1";
                break;
        case 2:
                result = "X1";
                break;
        case 3:
                result = "X2";
                break;
        case 4:
                result = "X3";
                break;
        case 5:
                result = "X4";
                break;
        case 6:
                result = "X4";
                break;
        }
        return result;
}

public static String changeTacTable(int roll){
        String result = null;
        switch (roll){
        case 6:
                result = "No";
                break;
        default:
                result = "Change";
        }
        return result;
}

//moves where will attack from
// TODO add comments on the number mean
public static String repositionDenizen(int roll){
        String result = null;
        switch (roll){
        case 1:
                result = "0-3-2";
                break;
```

```
                case 2:
                        result = "3-0-1";
                        break;
                case 3:
                        result = "2-1-0";
                        break;
                case 4:
                        result = "0-0-0";
                        break;
                case 5:
                        result = "2-3-1";
                        break;
                case 6:
                        result = "3-1-2";
                        break;
                }
                return result;
        }

//      check what search table to look through
        public static String seachTable(int selected, int roll){
                String result = null;
                switch (selected){
                case 1:
                        result = peerTable(roll);
                        break;
                case 2:
                        result = locateTable(roll);
                        break;
//                      add with magic
                /*case 3:
                        result = magicSightTable(roll);
                        break;
                case 4:
                        result = readRuneTable(roll);
                        break;*/
                }
                return result;
        }

        public static String peerTable(int roll){
                String result = null;
                switch (roll){
                case 1:
                        result = "Choice";
                        break;
                case 2:
                        result = "Clues-Paths";
                        break;
```

```
		case 3:
			result = "Hidden-Paths";
			break;
		case 4:
			result = "Hidden";
			break;
		case 5:
			result = "Clues";
			break;
		case 6:
			result = "No";
			break;
		}
		return result;
}
public static String locateTable(int roll){
		String result = null;
		switch (roll){
		case 1:
			result = "Choice";
			break;
		case 2:
			result = "Passages-Clues";
			break;
		case 3:
			result = "Passages";
			break;
		case 4:
			result = "Discover";
			break;
		case 5:
			result = "No";
			break;
		case 6:
			result = "No";
			break;
		}
		return result;
}

//TODO set to public with magic
private static String magicSightTable(int roll){
		String result = null;
		switch (roll){
		case 1:
			result = "Choice";
			break;
		case 2:
			result = "Counters";
```

```
                break;
            case 3:
                result = "Tresure";
                break;
            case 4:
                result = "Perceive";
                break;
            case 5:
                result = "Discover";
                break;
            case 6:
                result = "No";
                break;
            }
            return result;
        }

        //TODO set to public with magic
        private static String readRuneTable(int roll){
            String result = null;
            switch (roll){
            case 1:
                result = "Learn";
                break;
            case 2:
                result = "Learn";
                break;
            case 3:
                result = "Learn";
                break;
            case 4:
                result = "Awaken";
                break;
            case 5:
                result = "Curse";
                break;
            case 6:
                result = "No";
                break;
            }
            return result;
        }
}
```

Class: Pair.java

```
package utils;

import java.io.Serializable;

public class Pair<A, B> implements Serializable {
```

```
  private static final long serialVersionUID = -3022399295287625959L;
  private A first;
private B second;

// Pair Constructor
public Pair(A first, B second) {
  super();
  this.first = first;
  this.second = second;
}

// Return A Hashing Of The Data
public int hashCode() {
  int hashFirst = first != null ? first.hashCode() : 0;
  int hashSecond = second != null ? second.hashCode() : 0;

  return (hashFirst + hashSecond) * hashSecond + hashFirst;
}

// Returns True If 2 Pairs Are Equal
public boolean equals(Object other) {
  if (other instanceof Pair) {
        Pair otherPair = (Pair) other;
        return
        ((  this.first == otherPair.first ||
              ( this.first != null && otherPair.first != null &&
                this.first.equals(otherPair.first))) &&
         (    this.second == otherPair.second ||
              ( this.second != null && otherPair.second != null &&
                this.second.equals(otherPair.second))) );
  }

  return false;
}

// Returns The ToString() Of Both Objects Combined
public String toString()
{
       return "(" + first + ", " + second + ")";
}

public A getFirst() {
  return first;
}

public void setFirst(A first) {
  this.first = first;
}

public B getSecond() {
  return second;
}

public void setSecond(B second) {
  this.second = second;
}
}
```

Package: Views

Class: AddPlayerView.java

```
package views;

import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JTextField;

import models.characterModels.playerEnums.CharacterClass;

public class AddPlayerView extends FrameBase {

	/**
	 *
	 */
	private static final long serialVersionUID = -1991225057723904889L;

	public GameView parent;

	protected JLabel nameLabel;
	protected JLabel chooseLabel;
	protected JTextField nameField;
	protected JComboBox<CharacterClass> classSelecter;

	protected JButton startButton;
	protected JButton cancelButton;

	private Toolkit tk = Toolkit.getDefaultToolkit();

	public AddPlayerView (GameView parent) {
		super ();
		this.parent = parent;

		initWindow();
		initComponents();
		initListeners();
	}

	// Init The Window
	public void initWindow () {
		setSize(300,200);
		setLocation(((int)tk.getScreenSize().getWidth()/2) - 150,
((int)tk.getScreenSize().getHeight()/2) - 75);
		setVisible(true);
```

```
                setLayout(layout);
        }

        // Init All The Components
        public void initComponents () {
                nameField = new JTextField();
                nameField = new JTextField();
                nameLabel = new JLabel("Enter Player Name:");
                chooseLabel = new JLabel("Choose Your Class:");

                classSelecter = new JComboBox<CharacterClass>();
                classSelecter.setModel(new
DefaultComboBoxModel<CharacterClass>(CharacterClass.values()));

                startButton = new JButton("OK");
                cancelButton = new JButton("Cancel");

                addToFrame(this, nameLabel, layout, 0, 0, 1, 1);
                addToFrame(this, nameField, layout, 1, 0, 1, 1);
                addToFrame(this, chooseLabel, layout, 0, 1, 1, 1);
                addToFrame(this, classSelecter, layout, 1, 1, 1, 1);
                addToFrame(this, startButton, layout, 0, 2, 1, 1);
                addToFrame(this, cancelButton, layout, 1, 2, 1, 1);
        }

        // Initialize The Listeners
        public void initListeners () {
      cancelButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
                closeWindow();
        }
      });

      startButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
                parent.addPlayer(nameField.getText(),
classSelecter.getItemAt(classSelecter.getSelectedIndex()));
            closeWindow();
        }
      });
        }

        // Closes The Window When Called
        public void closeWindow () {
                dispose();
        }
}
```

Class: BoardView.java

```
package views;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;

import javax.imageio.ImageIO;
import javax.swing.JFrame;
import javax.swing.JPanel;

import models.BoardModels.Clearing;
import models.BoardModels.Dwelling;
import models.characterModels.playerEnums.CharacterClass;
import models.otherEntities.TreasureModel;

public class BoardView extends JPanel {

	private static final long serialVersionUID = -32551821833126394 41L;

	//Field declarations
	private Image img;
	private Toolkit tk = Toolkit.getDefaultToolkit();

	private ArrayList<Clearing> theButtons = new ArrayList<Clearing>();

	private Clearing cliff1 = new Clearing("cl1");
	private Clearing cliff2 = new Clearing("cl2");
	private Clearing cliff3 = new Clearing("cl3");
	private Clearing cliff4 = new Clearing("cl4");
	private Clearing cliff5 = new Clearing("cl5");
	private Clearing cliff6 = new Clearing("cl6");

	private Clearing evalley1 = new Clearing("ev1");
	private Clearing evalley2 = new Clearing("ev2");
	private Clearing evalley4 = new Clearing("ev4");
	private Clearing evalley5 = new Clearing("ev5");

	private Clearing hpass1 = new Clearing("hp1");
	private Clearing hpass2 = new Clearing("hp2");
	private Clearing hpass3 = new Clearing("hp3");
	private Clearing hpass4 = new Clearing("hp4");
```

```
private Clearing hpass5 = new Clearing("hp5");
private Clearing hpass6 = new Clearing("hp6");

private Clearing ledges1 = new Clearing("le1");
private Clearing ledges2 = new Clearing("le2");
private Clearing ledges3 = new Clearing("le3");
private Clearing ledges4 = new Clearing("le4");
private Clearing ledges5 = new Clearing("le5");
private Clearing ledges6 = new Clearing("le6");

private Clearing bland1 = new Clearing("bl1");
private Clearing bland2 = new Clearing("bl2");
private Clearing bland3 = new Clearing("bl3");
private Clearing bland4 = new Clearing("bl4");
private Clearing bland5 = new Clearing("bl5");
private Clearing bland6 = new Clearing("bl6");

private Clearing cavern1 = new Clearing("ca1");
private Clearing cavern2 = new Clearing("ca2");
private Clearing cavern3 = new Clearing("ca3");
private Clearing cavern4 = new Clearing("ca4");
private Clearing cavern5 = new Clearing("ca5");
private Clearing cavern6 = new Clearing("ca6");

private Clearing crag1 = new Clearing("cr1");
private Clearing crag2 = new Clearing("cr2");
private Clearing crag3 = new Clearing("cr3");
private Clearing crag4 = new Clearing("cr4");
private Clearing crag5 = new Clearing("cr5");
private Clearing crag6 = new Clearing("cr6");

private Clearing owoods2 = new Clearing("ow2");
private Clearing owoods4 = new Clearing("ow4");
private Clearing owoods5 = new Clearing("ow5");

private Clearing bvalley1 = new Clearing("bv1");
private Clearing bvalley2 = new Clearing("bv2");
private Clearing bvalley4 = new Clearing("bv4");
private Clearing bvalley5 = new Clearing("bv5");

private Clearing mountain1 = new Clearing("mo1");
private Clearing mountain2 = new Clearing("mo2");
private Clearing mountain3 = new Clearing("mo3");
private Clearing mountain4 = new Clearing("mo4");
private Clearing mountain5 = new Clearing("mo5");
private Clearing mountain6 = new Clearing("mo6");

private Clearing dvalley1 = new Clearing("dv1");
private Clearing dvalley2 = new Clearing("dv2");
```

```
private Clearing dvalley4 = new Clearing("dv4");
private Clearing dvalley5 = new Clearing("dv5");

private Clearing dwoods1 = new Clearing("dw1");
private Clearing dwoods2 = new Clearing("dw2");
private Clearing dwoods3 = new Clearing("dw3");
private Clearing dwoods4 = new Clearing("dw4");
private Clearing dwoods5 = new Clearing("dw5");
private Clearing dwoods6 = new Clearing("dw6");

private Clearing mwoods2 = new Clearing("mw2");
private Clearing mwoods4 = new Clearing("mw4");
private Clearing mwoods5 = new Clearing("mw5");

private Clearing caves1 = new Clearing("cv1");
private Clearing caves2 = new Clearing("cv2");
private Clearing caves3 = new Clearing("cv3");
private Clearing caves4 = new Clearing("cv4");
private Clearing caves5 = new Clearing("cv5");
private Clearing caves6 = new Clearing("cv6");

private Clearing pwoods2 = new Clearing("pw2");
private Clearing pwoods4 = new Clearing("pw4");
private Clearing pwoods5 = new Clearing("pw5");

private Clearing cvalley1 = new Clearing("cv1");
private Clearing cvalley2 = new Clearing("cv2");
private Clearing cvalley4 = new Clearing("cv4");
private Clearing cvalley5 = new Clearing("cv5");

private Clearing nwoods5 = new Clearing("nw5");
private Clearing nwoods2 = new Clearing("nw2");
private Clearing nwoods4 = new Clearing("nw4");

private Clearing ruins1 = new Clearing("ru1");
private Clearing ruins2 = new Clearing("ru2");
private Clearing ruins3 = new Clearing("ru3");
private Clearing ruins4 = new Clearing("ru4");
private Clearing ruins5 = new Clearing("ru5");
private Clearing ruins6 = new Clearing("ru6");

private Clearing avalley1 = new Clearing("av1");
private Clearing avalley2 = new Clearing("av2");
private Clearing avalley4 = new Clearing("av4");
private Clearing avalley5 = new Clearing("av5");

private Clearing lwoods2 = new Clearing("lw2");
private Clearing lwoods4 = new Clearing("lw4");
private Clearing lwoods5 = new Clearing("lw5");
```

```
	private Dwelling inn;

	private GameView parent;
	private JFrame hoverFrame = null;
	private HoverView hoverPanel;

	//Constructor for the BoardView
	public BoardView (GameView parent){
		init();
		this.parent = parent;

		// Now For The Inn
		try {
			Image innImage =
ImageIO.read(getClass().getResource("/dwellings_c/inn.gif"));
			inn = new Dwelling(dvalley4, innImage);
			inn.getClearingThisOn().addImageToList(inn.getImageRepresentation());
		} catch (Exception e) {
			e.printStackTrace();
		}
	}
	//Initialization method, sets up the board
	private void init(){
		try {
			img = ImageIO.read(getClass().getResource("/theMap3.gif"));
		} catch (IOException e) {
			e.printStackTrace();
		}

		setPreferredSize( new Dimension(1300,1486));
		setLayout(null);

		theButtons.add(cliff1);
		theButtons.add(cliff2);
		theButtons.add(cliff3);
		theButtons.add(cliff4);
		theButtons.add(cliff5);
		theButtons.add(cliff6);

		theButtons.add(evalley1);
		theButtons.add(evalley2);
		theButtons.add(evalley4);
		theButtons.add(evalley5);

		theButtons.add(hpass1);
		theButtons.add(hpass2);
		theButtons.add(hpass3);
		theButtons.add(hpass4);
```

```
theButtons.add(hpass5);
theButtons.add(hpass6);

theButtons.add(ledges1);
theButtons.add(ledges2);
theButtons.add(ledges3);
theButtons.add(ledges4);
theButtons.add(ledges5);
theButtons.add(ledges6);

theButtons.add(bland1);
theButtons.add(bland2);
theButtons.add(bland3);
theButtons.add(bland4);
theButtons.add(bland5);
theButtons.add(bland6);

theButtons.add(cavern1);
theButtons.add(cavern2);
theButtons.add(cavern3);
theButtons.add(cavern4);
theButtons.add(cavern5);
theButtons.add(cavern6);

theButtons.add(crag1);
theButtons.add(crag2);
theButtons.add(crag3);
theButtons.add(crag4);
theButtons.add(crag5);
theButtons.add(crag6);

theButtons.add(owoods2);
theButtons.add(owoods4);
theButtons.add(owoods5);

theButtons.add(bvalley1);
theButtons.add(bvalley2);
theButtons.add(bvalley4);
theButtons.add(bvalley5);

theButtons.add(mountain1);
theButtons.add(mountain2);
theButtons.add(mountain3);
theButtons.add(mountain4);
theButtons.add(mountain5);
theButtons.add(mountain6);

theButtons.add(dvalley1);
theButtons.add(dvalley2);
```

```
theButtons.add(dvalley4);
theButtons.add(dvalley5);

theButtons.add(dwoods1);
theButtons.add(dwoods2);
theButtons.add(dwoods3);
theButtons.add(dwoods4);
theButtons.add(dwoods5);
theButtons.add(dwoods6);

theButtons.add(mwoods2);
theButtons.add(mwoods4);
theButtons.add(mwoods5);

theButtons.add(caves1);
theButtons.add(caves2);
theButtons.add(caves3);
theButtons.add(caves4);
theButtons.add(caves5);
theButtons.add(caves6);

theButtons.add(pwoods2);
theButtons.add(pwoods4);
theButtons.add(pwoods5);

theButtons.add(cvalley1);
theButtons.add(cvalley2);
theButtons.add(cvalley4);
theButtons.add(cvalley5);

theButtons.add(nwoods5);
theButtons.add(nwoods2);
theButtons.add(nwoods4);

theButtons.add(ruins1);
theButtons.add(ruins2);
theButtons.add(ruins3);
theButtons.add(ruins4);
theButtons.add(ruins5);
theButtons.add(ruins6);

theButtons.add(avalley1);
theButtons.add(avalley2);
theButtons.add(avalley4);
theButtons.add(avalley5);

theButtons.add(lwoods2);
theButtons.add(lwoods4);
theButtons.add(lwoods5);
```

```
		//Adds the components to the board
		for(int i = 0; i < theButtons.size(); i++){
			add(theButtons.get(i).getButtonTiedToClearing());
		}

		initClearings();
		addClearingListeners();
	}

	// Adds All Of The Clearing Listeners
	private void addClearingListeners() {
		for (Clearing c: theButtons) {
			addListener(c);
		}
	}

	//Initializes the clearings
	private void initClearings() {
		cliff1.setLocation(419, 188);
		cliff2.setLocation(496, 204);
		cliff3.setLocation(473, 134);
		cliff4.setLocation(449, 62);
		cliff5.setLocation(527, 78);
		cliff6.setLocation(400, 120);

		cliff1.addToConnectedClearings(cliff1, evalley2);
		cliff2.addToConnectedClearings(ledges3, cliff3);
		cliff3.addToConnectedClearings(cliff6, cliff5, cliff2);
		cliff4.addConnectedClearing(cliff6);
		cliff5.addToConnectedClearings(cliff3, cliff2);
		cliff6.addToConnectedClearings(cliff1, cliff4, cliff3);

		evalley1.setLocation(255,267);
		evalley2.setLocation(355,260);
		evalley4.setLocation(350,355);
		evalley5.setLocation(250,370);

		evalley1.addConnect
edClearing(evalley4);
		evalley2.addToConnectedClearings(cliff1, evalley5);
		evalley4.addToConnectedClearings(ledges2, evalley1, bland2);
		evalley5.addToConnectedClearings(hpass6, evalley2);

		hpass1.setLocation(105,540);
		hpass2.setLocation(210,510);
		hpass3.setLocation(160,555);
		hpass4.setLocation(130,475);
		hpass5.setLocation(65,475);
```

```
hpass6.setLocation(188,435);

hpass1.addToConnectedClearings(hpass5, hpass4);
hpass2.addToConnectedClearings(hpass4, bland1);
hpass3.addToConnectedClearings(hpass6, cavern5);
hpass4.addToConnectedClearings(hpass1, hpass2);
hpass5.addConnectedClearing(hpass1);
hpass6.addToConnectedClearings(hpass3, evalley5);

ledges1.setLocation(572,321);
ledges2.setLocation(470,350);
ledges3.setLocation(515,300);
ledges4.setLocation(540,372);
ledges5.setLocation(545,445);
ledges6.setLocation(628,342);

ledges1.addToConnectedClearings(ledges6, ledges3, ledges4);
ledges2.addToConnectedClearings(evalley4, ledges5);
ledges3.addToConnectedClearings(ledges6, ledges1, cliff2);
ledges4.addToConnectedClearings(bland4, ledges1, ledges6);
ledges5.addToConnectedClearings(ledges2, owoods2);
ledges6.addToConnectedClearings(ledges1, ledges3, ledges4);

bland1.setLocation(285,570);
bland2.setLocation(385,470);
bland3.setLocation(315,500);
bland4.setLocation(445,613);
bland5.setLocation(395,590);
bland6.setLocation(350,558);

bland1.addToConnectedClearings(hpass2, bland6);
bland2.addToConnectedClearings(owoods2, bland3, evalley4);
bland3.addToConnectedClearings(bland2, bland6, bland5);
bland4.addToConnectedClearings(ledges4, bland5, bland6);
bland5.addToConnectedClearings(bland3, bland4, cavern2);
bland6.addToConnectedClearings(bland1, bland3, bland4);

cavern1.setLocation(290,742);
cavern2.setLocation(262,665);
cavern3.setLocation(222,708);
cavern4.setLocation(155,786);
cavern5.setLocation(185,655);
cavern6.setLocation(210,768);

cavern1.addToConnectedClearings(cavern3, cavern4, bvalley4);
cavern2.addToConnectedClearings(cavern3, bland5);
cavern3.addToConnectedClearings(cavern1, cavern2, cavern5, cavern6);
cavern4.addToConnectedClearings(cavern1, cavern5, cavern6);
cavern5.addToConnectedClearings(cavern3, cavern4, hpass3);
```

```
cavern6.addToConnectedClearings(cavern3, cavern4);

crag1.setLocation(764,345);
crag2.setLocation(800,500);
crag3.setLocation(828,441);
crag4.setLocation(730,403);
crag5.setLocation(748,464);
crag6.setLocation(816,382);

crag1.addToConnectedClearings(crag4, crag6);
crag2.addToConnectedClearings(crag5, crag3, dwoods1);
crag3.addToConnectedClearings(crag2, crag5, crag6);
crag4.addToConnectedClearings(crag1, crag4);
crag5.addToConnectedClearings(crag3, crag2);
crag6.addToConnectedClearings(crag3, crag4, crag1);

owoods2.setLocation(561,565);
owoods4.setLocation(681,621);
owoods5.setLocation(596,668);

owoods2.addToConnectedClearings(owoods4, bland2, ledges5);
owoods4.addToConnectedClearings(owoods2, dwoods1);
owoods5.addToConnectedClearings(bvalley1, mwoods5);

bvalley1.setLocation(500,719);
bvalley2.setLocation(470,851);
bvalley4.setLocation(375,802);
bvalley5.setLocation(423,709);

bvalley1.addToConnectedClearings(bvalley4, owoods5);
bvalley2.addToConnectedClearings(bvalley5, caves2);
bvalley4.addToConnectedClearings(cavern1, mountain5);
bvalley5.addToConnectedClearings(bland1, bland2);

mountain1.setLocation(248, 999);
mountain2.setLocation(306, 1040);
mountain3.setLocation(319, 973);
mountain4.setLocation(200, 940);
mountain5.setLocation(337, 900);
mountain6.setLocation(263, 899);

mountain1.addConnectedClearing(mountain3);
mountain2.addToConnectedClearings(mountain4, mountain5, pwoods4);
mountain3.addToConnectedClearings(mountain1, mountain6);
mountain4.addToConnectedClearings(mountain6, mountain2);
mountain5.addToConnectedClearings(mountain6, mountain2, bvalley4);
mountain6.addToConnectedClearings(mountain4, mountain5, mountain3);

dvalley1.setLocation(1037, 560);
```

```
dvalley2.setLocation(1091, 508);
dvalley4.setLocation(1003, 450);
dvalley5.setLocation(966, 543);

dvalley1.addToConnectedClearings(cvalley1, dvalley4);
dvalley2.addToConnectedClearings(dvalley5);
dvalley4.addToConnectedClearings(dvalley1);

dwoods1.setLocation(782,620);
dwoods2.setLocation(918,646);
dwoods3.setLocation(895,730);
dwoods4.setLocation(760,691);
dwoods5.setLocation(808,739);
dwoods6.setLocation(846,680);

dwoods1.addToConnectedClearings(owoods4, crag2, dwoods6, dwoods4);
dwoods2.addToConnectedClearings(dwoods3, cvalley2, dvalley5);
dwoods3.addToConnectedClearings(dwoods6, dwoods5, dwoods2);
dwoods4.addToConnectedClearings(dwoods1, dwoods5, dwoods6);
dwoods5.addToConnectedClearings(mwoods5, dwoods4, dwoods3);
dwoods6.addToConnectedClearings(dwoods1, dwoods4, dwoods3);

mwoods2.setLocation(730,887);
mwoods4.setLocation(634,898);
mwoods5.setLocation(695,776);

mwoods2.addToConnectedClearings(nwoods5, mwoods4, ruins5);
mwoods4.addToConnectedClearings(caves5, mwoods2);
mwoods5.addToConnectedClearings(owoods5, dwoods5);

caves1.setLocation(460,1071);
caves2.setLocation(493,945);
caves3.setLocation(518,1004);
caves4.setLocation(438,1012);
caves5.setLocation(564,959);
caves6.setLocation(577,1067);

caves1.addToConnectedClearings(caves6, pwoods5);
caves2.addToConnectedClearings(caves3, caves4, bvalley2);
caves3.addToConnectedClearings(caves2, caves5);
caves4.addToConnectedClearings(caves2, caves6);
caves5.addToConnectedClearings(mwoods5, caves3);
caves6.addToConnectedClearings(caves4, caves1);

pwoods2.setLocation(274, 1203);
pwoods4.setLocation(320, 1126);
pwoods5.setLocation(412, 1175);

pwoods2.addToConnectedClearings(pwoods4);
```

```
pwoods4.addToConnectedClearings(pwoods2, mountain2);
pwoods5.addToConnectedClearings(caves1);

cvalley1.setLocation(1063, 650);
cvalley2.setLocation(1010, 700);
cvalley4.setLocation(1094, 759);
cvalley5.setLocation(1135, 667);

cvalley1.addToConnectedClearings(dvalley1, cvalley4);
cvalley2.addToConnectedClearings(dwoods2, cvalley5);
cvalley4.addToConnectedClearings(nwoods5, cvalley1);
cvalley5.addConnectedClearing(cvalley2);

nwoods2.setLocation(970,945);
nwoods4.setLocation(876,959);
nwoods5.setLocation(910,858);

nwoods2.addToConnectedClearings(nwoods4, avalley5);
nwoods4.addToConnectedClearings(ruins1, nwoods2);
nwoods5.addToConnectedClearings(cvalley4, mwoods2);

ruins1.setLocation(801,1031);
ruins2.setLocation(815,1128);
ruins3.setLocation(699,1129);
ruins4.setLocation(730,1060);
ruins5.setLocation(700,1000);
ruins6.setLocation(760,1120);

ruins1.addToConnectedClearings(ruins5, nwoods4, ruins2, ruins4);
ruins2.addToConnectedClearings(ruins1, lwoods4, avalley1);
ruins3.addToConnectedClearings(ruins6, ruins5);
ruins4.addToConnectedClearings(ruins1, ruins6);
ruins5.addToConnectedClearings(ruins3, mwoods2, ruins1);
ruins6.addToConnectedClearings(ruins3, ruins4);

avalley1.setLocation(915,1112);
avalley2.setLocation(930,1180);
avalley4.setLocation(1028,1140);
avalley5.setLocation(968,1058);

avalley1.addToConnectedClearings(ruins2, avalley4);
avalley2.addToConnectedClearings(lwoods5, avalley5);
avalley4.addConnectedClearing(avalley1);
avalley5.addToConnectedClearings(avalley2, nwoods2);

lwoods2.setLocation(803,1330);
lwoods4.setLocation(800,1235);
lwoods5.setLocation(883,1286);
```

```
                lwoods2.addConnectedClearing(lwoods4);
                lwoods4.addToConnectedClearings(lwoods2, ruins2);
                lwoods5.addConnectedClearing(avalley2);

                // Temp
                cavern1.addTreasures(new TreasureModel(true), new TreasureModel(false));
        }

        //Adds a listener to the clearing that is passed to this function
        private void addListener (final Clearing c) {
                c.getButtonTiedToClearing().addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
                if (parent.getCurrentPlayer().isMoving() &&
parent.getCurrentPlayer().getCurrentClearing().isVaildMove(c)){
                        parent.getCurrentPlayer().moveToClearing(c);

                        // If Player Has No More Movements Then Stop Them
                        if (parent.getCurrentPlayer().getAvailableActions() > 0) {
                                c.highlightConnectedClearings();
                        } else {
                                parent.getCurrentPlayer().setMoving(false);
                        }
                }
        }
      });

                // Display The Images Here
                c.getButtonTiedToClearing().addMouseListener(new MouseAdapter() {
                        @Override
                        public void mouseEntered(MouseEvent e) {
                                handleImageEnter(c);
                        }
                });
        }

        // Handles The Mouse Entering The Image
        private void handleImageEnter (Clearing c) {

                if(c.getImageEnitiesOnThis().isEmpty() || c.getImageEnitiesOnThis().size() == 1){
                        return;
                }
                if(hoverFrame!=null){
                        hoverFrame.dispose();
                }
                hoverFrame = new JFrame();
                hoverPanel = new HoverView(c);
                hoverFrame.setSize(160, 250);
                hoverFrame.setLocation(tk.getScreenSize().width/2 - 300, 300);
                hoverFrame.setVisible(true);
```

```
            hoverFrame.add(hoverPanel);
        }

        //Overrides the paint component method of jPanel
        @Override
        public void paintComponent(Graphics page)
        {
           super.paintComponent(page);
           page.drawImage(img, 0, 0, this);
        }

        //returns the default clearing of a class (always the inn)
        public Clearing getDefaultClearingForClass (CharacterClass c) {
            switch (c) {
            case SWORDSMAN: return inn.getClearingThisOn();
            case AMAZON: return inn.getClearingThisOn();
            case BLACKNIGHT: return inn.getClearingThisOn();
            case CAPTAIN: return inn.getClearingThisOn();
            case DWARF: return inn.getClearingThisOn();
            case ELF: return inn.getClearingThisOn();
            default: return avalley2;
            }
        }
}
```

Class: CardView.java

```
package views;

import java.awt.Graphics;
import java.awt.Image;
import java.awt.Toolkit;

import javax.swing.JPanel;

import models.characterModels.playerEnums.CharacterClass;

public class CardView extends JPanel {

        /**
         *
         */
        private static final long serialVersionUID = 3724840708083857909L;
        private Toolkit tk = Toolkit.getDefaultToolkit();
        private Image img;

        //constructor for cardview, displays the detailed character stats of the currently selected players
class
        public CardView(String valueAt) {
            setSize(600,600);
```

```
                setLocation(((int)tk.getScreenSize().getWidth()/2) - 300,
((int)tk.getScreenSize().getHeight()/2) - 300);
                setVisible(true);
                CharacterClass charClass = CharacterClass.valueOf(valueAt);
                img = charClass.getDetailImage();
        }

        //Overrides the paintComponent method in JPanel
        @Override
        public void paintComponent(Graphics page)
        {
           super.paintComponent(page);
           page.drawImage(img, 0, 0, this);
        }
}
```

Class: CombatView.java

```
package views;

import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

import javax.swing.DefaultComboBoxModel;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JScrollPane;

import models.characterModels.PlayerBase;
import models.characterModels.playerEnums.Attacks;
import models.chitModels.WeaponChit;
import controller.CombatPVPHandler;

public class CombatView extends FrameBase {

   /**
         *
         */
        private static final long serialVersionUID = -6115145956127664758L;
   // Variables declaration
   private javax.swing.JButton chargeButton;
   private javax.swing.JButton dodgeButton;
   private javax.swing.JButton duckButton;
   private javax.swing.JButton runButton;
   private javax.swing.JButton alertButton;
   private javax.swing.JButton activateButton;
   private javax.swing.JButton abandonButton;
   private javax.swing.JButton nextButton;
```

```
    private javax.swing.JButton endButton;
    private javax.swing.JButton resetButton;
    private javax.swing.JButton thrustButton;
    private javax.swing.JButton swingButton;
    private javax.swing.JButton smashButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private JLabel targetPlayerLabel;
    private JLabel weaponLabel;
    private JLabel weaponHarmLabel;
    private javax.swing.JLabel smashShield;
    private javax.swing.JLabel swindShield;
    private javax.swing.JLabel thrustShield;
    private javax.swing.JLabel suitOfArmor;
    private javax.swing.JLabel breastPlate;
    private javax.swing.JLabel helmet;
    private javax.swing.JTextArea textArea;
    private JScrollPane textContainer;
    private JComboBox<PlayerBase> playersCanAttack;

    // Setup The Combat Handler For The View
    private CombatPVPHandler combatHandler;
    private ArrayList<PlayerBase> combatingPlayers;
    private ArrayList<PlayerBase> targetPlayers;
    private Toolkit tk = Toolkit.getDefaultToolkit();

    // Constructor For This
    public CombatView(ArrayList<PlayerBase> combatingPlayers) {
        this.combatingPlayers = combatingPlayers;
        combatHandler = new CombatPVPHandler(combatingPlayers, this);

        initComponents();
       initWindow();
       addAllComponents();
       update();
    }

    // Initialize The Window
    public void initWindow () {
                setSize(760,300);
                setLocation(((int)tk.getScreenSize().getWidth()/2) - 300,
((int)tk.getScreenSize().getHeight()/2) - 300);
                setLayout(layout);
                setName ("Combat Screen");
                setVisible(true);
    }

    // Adds All Of The Components To The Window
    public void addAllComponents () {
```

```
        // First Is The Text Area
        addToFrame(this, textContainer, layout, 0, 0, 5, 5);

        // Add In All Of The Attack Buttons Buttons
        addToFrame(this, jLabel2, layout, 5, 0, 1, 1);
        addToFrame(this, smashButton, layout, 6, 0, 1, 1);
        addToFrame(this, thrustButton, layout, 7, 0, 1, 1);
        addToFrame(this, swingButton, layout, 8, 0, 1, 1);

        // Weapon Information
        addToFrame(this, weaponLabel, layout, 5, 1, 1, 1);
        addToFrame(this, weaponHarmLabel, layout, 6, 1, 1, 1);

        // Attackable Players Window
        addToFrame(this, targetPlayerLabel, layout, 5, 2, 1, 1);
        addToFrame(this, playersCanAttack, layout, 6, 2, 3, 1);

        // More Buttons
        addToFrame (this, nextButton, layout, 5, 4, 2, 1);
        addToFrame (this, runButton, layout, 7, 4, 2, 1);
}

private void initComponents() {

    textArea = new javax.swing.JTextArea();
    textContainer = new JScrollPane(textArea);

    // Buttons
    thrustButton = new javax.swing.JButton();
    swingButton = new javax.swing.JButton();
    smashButton = new javax.swing.JButton();
    chargeButton = new javax.swing.JButton();
    dodgeButton = new javax.swing.JButton();
    duckButton = new javax.swing.JButton();
    jLabel1 = new javax.swing.JLabel();
    jLabel2 = new javax.swing.JLabel();
    runButton = new javax.swing.JButton();
    alertButton = new javax.swing.JButton();
    activateButton = new javax.swing.JButton();
    abandonButton = new javax.swing.JButton();
    nextButton = new javax.swing.JButton();
    endButton = new javax.swing.JButton();

    // Labels
    smashShield = new javax.swing.JLabel();
    swindShield = new javax.swing.JLabel();
    thrustShield = new javax.swing.JLabel();
    suitOfArmor = new javax.swing.JLabel();
```

```
        breastPlate = new javax.swing.JLabel();
        helmet = new javax.swing.JLabel();
        resetButton = new javax.swing.JButton();
        targetPlayerLabel = new JLabel("Target:");
        weaponLabel = new JLabel();
        weaponHarmLabel = new JLabel();
        playersCanAttack = new JComboBox<>();

        // Setup The Listeners For This View
        setupListeners();

        // Setup The Scroll Pane For The Text Field
        textArea.setColumns(20);
        textArea.setRows(5);

textContainer.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_NEVER);

textContainer.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);

        // Set the text of all the components
        thrustButton.setText("Thrust");
        swingButton.setText("Swing");
        smashButton.setText("Smash");
        chargeButton.setText("Charge");
        dodgeButton.setText("Dodge");
        duckButton.setText("Duck");
        jLabel1.setText("Defences:");
        jLabel2.setText("Attacks:");
        runButton.setText("Run");
        alertButton.setText("Alert Wepon/Chit");
        activateButton.setText("Activate/Inactivate");
        abandonButton.setText("Abandon Belongings");
        nextButton.setText("Move To Next Attacker");
        endButton.setText("End");
        smashShield.setText("Protects Against Smash");
        swindShield.setText("Protects Against Swing");
        thrustShield.setText("Protects Against Thrust");
        suitOfArmor.setText("Suit of Armor");
        breastPlate.setText("Breastplate");
        helmet.setText("Helmet");
        resetButton.setText("Reset");
    }

    // Set up all the action listeners
        private void setupListeners() {
                thrustButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            combatHandler.setCurrentAttack(Attacks.THRUST);
            nextButton.setEnabled(true);
```

```
        println("Setting Current Attack To " + Attacks.THRUST);
    }
});
swingButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        combatHandler.setCurrentAttack(Attacks.SWING);
        nextButton.setEnabled(true);
        println("Setting Current Attack To " + Attacks.SWING);
    }
});
smashButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        combatHandler.setCurrentAttack(Attacks.SMASH);
        nextButton.setEnabled(true);
        println("Setting Current Attack To " + Attacks.SMASH);
    }
});
dodgeButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        System.out.println("dodge pressed");
    }
});
duckButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        System.out.println("duck pressed");
    }
});
runButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        System.out.println("run pressed");
        dispose();
    }
});

nextButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        System.out.println("next pressed");
        combatHandler.setCurrentDefender(targetPlayers.get(playersCanAttack.getSelectedIndex()));

        // Display For Display Purposes
        println ("Attack Submitted:");
        println ("  --- Attacker: " + combatHandler.getCurrentAttacker().getName());
        println ("  --- Defender: " + combatHandler.getCurrentDefender().getName());
        println ("");

        // Go To The Next Attack
        combatHandler.setNextAttacker();
        nextButton.setEnabled(false);
        update();
```

```
        }
    });
    endButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            System.out.println("end pressed");
        }
    });
    alertButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            System.out.println("alert pressed");
        }
    });
    activateButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            System.out.println("activate pressed");
        }
    });
    abandonButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            System.out.println("abandon pressed");
        }
    });
    resetButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
            System.out.println("reset pressed");
        }
    });
        }

// Updates This Window With Need Material
private void update () {
        // Get The Combobox ready
        targetPlayers = getPlayersToAttack();
    PlayerBase[] attackables = new PlayerBase[targetPlayers.size()];
    attackables = targetPlayers.toArray(attackables);
    DefaultComboBoxModel<PlayerBase> model = new DefaultComboBoxModel<>( attackables );
    playersCanAttack.setModel(model);

    // Set The Next Player To Disabled
    nextButton.setText((combatHandler.getReadyPlayerNum() == combatingPlayers.size() - 1)
                                        ? "Start Combat" : "Move To Next Attacker");

    // Player Weapon Display
    WeaponChit equipWeapon = combatHandler.getCurrentAttacker().getWeapon();
    weaponLabel.setText("Equipped Weapon: " + equipWeapon.getWeaponName());
    weaponHarmLabel.setText("Harm Level: " + equipWeapon.getWeaponDamage());
}

// Get All Of The Players To Attack
```

```
    private ArrayList<PlayerBase> getPlayersToAttack () {
        ArrayList<PlayerBase> returnVal = new ArrayList<>();
        for (PlayerBase p: combatingPlayers) {
            if (p != combatHandler.getCurrentAttacker()) {
                returnVal.add(p);
            }
        }
        return returnVal;
    }

    //Used to set the armor labels
    public void setArmor(String suit, String breast, String newHelmet){
        suitOfArmor.setText(suit);
        breastPlate.setText(breast);
        helmet.setText(newHelmet);
    }

    //used to print a line to the text area
    public void println(String theLine){
        textArea.append(theLine + "\n");
    }
}
```

Class: FrameBase.java

```
package views;

import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;

import javax.swing.JComponent;
import javax.swing.JFrame;

public class FrameBase extends JFrame {

        /**
         *
         */
        private static final long serialVersionUID = -331601846460123149L;
        protected GridBagLayout layout = new GridBagLayout();
        protected GridBagConstraints layoutConstraints = new GridBagConstraints();

        //A method for adding other jFrames to this FrameBase
        protected void addToFrame (JFrame connectToFrame, JComponent theComponent,
GridBagLayout layout, int x, int y, int gridWidth, int gridHeight){
            layoutConstraints.gridx = x;
            layoutConstraints.gridy = y;
            layoutConstraints.gridwidth = gridWidth;
            layoutConstraints.gridheight = gridHeight;
```

```
                layoutConstraints.fill = GridBagConstraints.BOTH;
                layoutConstraints.insets = new Insets(10, 5, 10, 5);
                layoutConstraints.anchor = GridBagConstraints.NORTHWEST;
                layoutConstraints.weightx = 1.0;
                layoutConstraints.weighty = 1.0;
                layout.setConstraints(theComponent, layoutConstraints);
                connectToFrame.add(theComponent);
        }
}
```

Class: GameView.java

```
package views;

import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ScrollPaneConstants;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;

import models.characterModels.PlayerBase;
import models.characterModels.playerEnums.CharacterClass;
import controller.clientController;

public class GameView extends FrameBase {

        private static final long serialVersionUID = 1789113344181363284L;

        //Field declarations
        private Toolkit tk = Toolkit.getDefaultToolkit();

        private clientController theClient;

        private JMenuBar menuBar = new JMenuBar();
```

```
	private JScrollPane scrollPane;

	private JPanel mainPanel;
	private BoardView theBoard;

	private GridBagLayout layout = new GridBagLayout();
	private GridBagConstraints layoutConstraints = new GridBagConstraints();

	private JScrollPane theBoardScroller;

	private PlayerListView thePlayerList;
	private PlayerControllView thePlayerButtons;

	private CardView theCard;

	private JFrame cardViewer;

	//Constructor for gameView
	public GameView(){
		init();
	}

	// Initialization Method
	public void init(){

		theClient = new clientController(this);

		mainPanel = new JPanel();
		scrollPane = new JScrollPane(mainPanel);


scrollPane.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED
);


scrollPane.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_AS_NE
EDED);
		mainPanel.setLayout(layout);
		setSize((int)tk.getScreenSize().getWidth(),(int)tk.getScreenSize().getHeight()-40);
		setVisible(true);
		setTitle("Magic Realm");
		setJMenuBar(menuBar);
		add(scrollPane);

		JMenu fileMenu = new JMenu("File");

		menuBar.add(fileMenu);
```

```
        JMenuItem newAction = new JMenuItem("New Game");
JMenuItem exitAction = new JMenuItem("Exit");

fileMenu.add(newAction);
fileMenu.add(exitAction);

//Action listeners
newAction.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent arg0) {
      startGame();
   }
});

exitAction.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent arg0) {
      exitGame();
   }
});

addWindowListener(new WindowAdapter(){
   public void windowClosing(WindowEvent e){
        exitGame();
   }
});

   }

   //update method resizes the screen to get it to repaint
   private void update(){
        setSize((int)tk.getScreenSize().getWidth(),(int)tk.getScreenSize().getHeight()-20);
        setSize((int)tk.getScreenSize().getWidth(),(int)tk.getScreenSize().getHeight()-40);
   }

   //Dispalys the gameButtons and user interface
   private void showGameButtons(){

        thePlayerList = new PlayerListView(this);
        addToGrid(thePlayerList, 0, 0, 1, 1);

        thePlayerButtons = new PlayerControllView(this);
        addToGrid(thePlayerButtons, 0, 1, 1, 2);

        //Action listeners
thePlayerList.getAddPlayerButton().addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      addPlayerMenu();
   }
});
```

```
        thePlayerList.getjButton2().addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                removePlayer();
            }
        });

        thePlayerList.getjTable2().getSelectionModel().addListSelectionListener(new
ListSelectionListener(){
            @Override
            public void valueChanged(ListSelectionEvent event){
                    if(thePlayerList.getjTable2().getSelectedRow()>-1){
                            setPlayerInterface(thePlayerList.getjTable2().getSelectedRow());
                    }
            }
        });

        thePlayerButtons.getjButton1().addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                showCard();
            }
        });

        thePlayerList.getStartGameButton().addActionListener(new ActionListener() {
            @Override
                            public void actionPerformed(ActionEvent arg0) {
                    handleStartGame();
                            }
                    });
            }

            //happens when you press the start game button, initlizes the game
            private void handleStartGame () {
                    System.out.println("Start Game Pressed");

                    theClient.startGame();
                    thePlayerList.updateTable();
                    thePlayerButtons.massSetButtonState(true);
                    thePlayerList.getStartGameButton().setEnabled(false);
                    JOptionPane.showMessageDialog(this, "The Game Has Started!");
            }

            //Shows the player's detailed character sheet
            private void showCard() {
                    if(thePlayerList.getjTable2().getSelectedRow() == -1){
                            return;
                    }
                    cardViewer = new JFrame();
                    cardViewer.setSize(760,630);
                    cardViewer.setLocation(((int)tk.getScreenSize().getWidth()/2) - 300,
```

```
((int)tk.getScreenSize().getHeight()/2) - 300);
                cardViewer.setVisible(true);
                theCard = new CardView((String)
thePlayerList.getjTable2().getValueAt(thePlayerList.getjTable2().getSelectedRow(), 0));
                cardViewer.add(theCard);
        }

        //changes the ui to show details of the selected player
        private void setPlayerInterface(int selectedRow) {
                thePlayerButtons.getPlayerClassLabel().setText((String)
thePlayerList.getjTable2().getValueAt(selectedRow, 0));
                thePlayerButtons.getPlayerDisplayLabel().setText((String)
thePlayerList.getjTable2().getValueAt(selectedRow, 1));
        }

        //removes the selected player for the player list and from the controller class
        private void removePlayer() {

                if(thePlayerList.getjTable2().getSelectedRow()==-1){
                        return;
                }

                int selectedOption = JOptionPane.showConfirmDialog(null,
           "Are you sure that you want to remove this player?",
           "Choose",
           JOptionPane.YES_NO_OPTION);
                if (selectedOption == JOptionPane.NO_OPTION) {
                        return;
                }

                String playerToRemoveName = (String)
thePlayerList.getjTable2().getModel().getValueAt(thePlayerList.getjTable2().getSelectedRow(),1);

                thePlayerList.removePlayer();
                theClient.removePlayer(playerToRemoveName);
                thePlayerList.update();
        }

        //Displays and initializes the board in the game view
        private void showBoard(){
                theBoard = new BoardView(this);
                theBoardScroller = new JScrollPane(theBoard);


theBoardScroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWA
YS);


theBoardScroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_
```

```
ALWAYS);
                theBoardScroller.getVerticalScrollBar().setUnitIncrement(20);
                addToGrid(theBoardScroller, 1, 0, 4, 4);
                theBoardScroller.setPreferredSize(new
Dimension(((int)tk.getScreenSize().getWidth()/2), ((int)tk.getScreenSize().getHeight()/2)));
        }

        //opens the addplayer menu that letss the user select their name and class
        private void addPlayerMenu(){
                new AddPlayerView(this);
        }

        //Adds the new palyer to the controller and the UI list
        public void addPlayer(String playerName, CharacterClass playerClass){
                theClient.addPlayer(playerClass, playerName);
                thePlayerList.addPlayer(playerName, playerClass.name());
                setPlayerInterface(thePlayerList.getjTable2().getRowCount()-1);
        }

        //closes the game and disposes of all components
        private void exitGame(){
                System.out.println("Exiting");
                dispose();
                System.exit(0);
        }

        // Starts The Game
        private void startGame(){
                showGameButtons();
                showBoard();
                update();
        }

        //updates a player given their name
        public void updatePlayer (String name, CharacterClass charClass) {
                PlayerBase p = theClient.getPlayer(name, charClass);

                if (p != null)
                        thePlayerButtons.update(p);
        }

        // Update The Table On Call
        public void updateRecordTable () {
                thePlayerButtons.update(getCurrentPlayer());
        }

        /*------------------------------ Getters And Setters ---------------------------*/
        public BoardView getBoardView () {
                return theBoard;
```

```
	}

	public PlayerBase getCurrentPlayer () {
		return theClient.getCurrentPlayer();
	}

	public clientController getGameController () {
		return theClient;
	}

	public PlayerListView getPlayerList() {
		return thePlayerList;
	}

	//checks if the game has started or not
	public boolean hasGameStarted() {
		return theClient.isGameStarted();
	}

	// Sets The Grid Location Based On The Paramenters Given
	private void addToGrid(JComponent theComponent, int x, int y, int gridWidth, int gridHeight) {
		addToGrid(mainPanel, theComponent, x, y, gridWidth, gridHeight);
	}

	//used to add components to this frame in specified locations
	private void addToGrid(JPanel connectToFrame, JComponent theComponent, int x, int y, int
gridWidth, int gridHeight) {
		layoutConstraints.gridx = x;
		layoutConstraints.gridy = y;
		layoutConstraints.gridwidth = gridWidth;
		layoutConstraints.gridheight = gridHeight;
		layoutConstraints.fill = GridBagConstraints.BOTH;
		layoutConstraints.insets = new Insets(10, 5, 10, 5);
		layoutConstraints.anchor = GridBagConstraints.NORTHWEST;
		layoutConstraints.weightx = 1.0;
		layoutConstraints.weighty = 1.0;
		layout.setConstraints(theComponent, layoutConstraints);
		mainPanel.add(theComponent);
	}
}
```

Class: HoverView.java

```
package views;

import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
```

```
import javax.swing.ImageIcon;

import models.BoardModels.Clearing;

public class HoverView extends javax.swing.JPanel {

	/**
	 *
	 */
	private static final long serialVersionUID = 38488765287041760 24L;

	private ArrayList<Image> imageList;

	private int currentSelection;

   // Variables declaration
   private javax.swing.JButton backButton;
   private javax.swing.JButton nextButton;
   private javax.swing.JLabel imageLabel;
   private javax.swing.JLabel counterLabel;
   // End of variables declaration

   //Constructor for the hoverview
	public HoverView(Clearing c) {
		initComponents();
		imageList = c.getImageEnitiesOnThis();
		currentSelection = 1;
		if(imageList!=null && imageList.size()>0){
			setLabel(imageList.get(0));
			setCounter(imageList.size(), currentSelection);
		}
	}

	//changes the numbers displayed in the item counter at the top
	private void setCounter(int size, int current){
		counterLabel.setText(current + "/" + size);
	}

	//used to change the image displayed in this view
	private void setLabel(Image image){
		ImageIcon icon = new ImageIcon(image);
		imageLabel.setIcon(icon);
	}

	//initializes this view
	private void initComponents() {

      imageLabel = new javax.swing.JLabel();
      counterLabel = new javax.swing.JLabel();
```

```
        backButton = new javax.swing.JButton();
        nextButton = new javax.swing.JButton();


        //Action listeners
        backButton.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent arg0) {
              back();
           }
        });

        nextButton.addActionListener(new ActionListener() {
           public void actionPerformed(ActionEvent arg0) {
                    next();
           }
        });

        //Set text for the components in this view
        imageLabel.setText("");
        counterLabel.setText("");
        backButton.setText("Back");
        nextButton.setText("Next");

        //The layout
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(
           layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
           .addGroup(layout.createSequentialGroup()
              .addContainerGap()
              .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                 .addComponent(imageLabel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                 .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                       .addGroup(layout.createSequentialGroup()
                          .addComponent(backButton)
                          .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                          .addComponent(nextButton))
                       .addComponent(counterLabel, javax.swing.GroupLayout.PREFERRED_SIZE, 34,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addGap(0, 0, Short.MAX_VALUE)))
              .addContainerGap())
        );
        layout.setVerticalGroup(
           layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
           .addGroup(layout.createSequentialGroup()
              .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
```

```
            .addComponent(counterLabel, javax.swing.GroupLayout.PREFERRED_SIZE, 24,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(imageLabel, javax.swing.GroupLayout.PREFERRED_SIZE, 121,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
               .addComponent(backButton)
               .addComponent(nextButton)))
      );
   }

        //done when pressing the next button
        private void next() {
                if(currentSelection < imageList.size()){
                        currentSelection++;
                        setLabel(imageList.get(currentSelection-1));
                        setCounter(imageList.size(),currentSelection);
                }
        }

        //done when pressing the back button
        private void back() {
                if(currentSelection > 1){
                        currentSelection--;
                        setLabel(imageList.get(currentSelection-1));
                        setCounter(imageList.size(),currentSelection);
                }
        }

}
```

Class: PlayerControlView.java

```
package views;

import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;

import models.characterModels.PlayerBase;
import models.otherEntities.TreasureModel;

public class PlayerControllView extends javax.swing.JPanel {
```

```
        // Variables declaration
    private javax.swing.JButton jButton1;
    private javax.swing.JButton sendTurnButton;
    private javax.swing.JButton cancelActionButton;
    private javax.swing.JButton hideButton;
    private javax.swing.JButton moveButton;
    private javax.swing.JButton searchButton;
    private javax.swing.JButton restButton;
    private javax.swing.JButton tradeButton;
    private javax.swing.JLabel playerFameLabel;
    private javax.swing.JLabel playerNotirityLabel;
    private javax.swing.JLabel playerVulnerLabel;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel18;
    private javax.swing.JLabel playerLabel;
    private javax.swing.JLabel playerClassDisplayLabel;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel classLabel;
    private javax.swing.JLabel playerDisplayLabel;
    private javax.swing.JLabel playerGoldLabel;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTabbedPane jTabbedPane1;
    private javax.swing.JTabbedPane jTabbedPane4;
    private javax.swing.JTabbedPane jTabbedPane5;
    private javax.swing.JTabbedPane jTabbedPane6;
    private javax.swing.JTable playerRecord;
        private static final long serialVersionUID = 1336340316590856087L;

        private GameView parent;
        private javax.swing.JTable jTable2;
        private JScrollPane jScrollPane3;
        private JTable jTable3;
        private JScrollPane jScrollPane4;

        private Toolkit tk = Toolkit.getDefaultToolkit();

        //constructor for the playerControllView
    public PlayerControllView(GameView parentView) {
       initComponents();
       massSetButtonState(false);
       parent = parentView;
```

```
}

//initializes the components in this view
private void initComponents() {

    setjButton1(new javax.swing.JButton());
    playerLabel = new javax.swing.JLabel();
    jLabel3 = new javax.swing.JLabel();
    jLabel4 = new javax.swing.JLabel();
    jLabel5 = new javax.swing.JLabel();
    jLabel6 = new javax.swing.JLabel();
    setjLabel8(new javax.swing.JLabel());
    setPlayerGoldLabel(new javax.swing.JLabel());
    playerFameLabel = new javax.swing.JLabel();
    playerNotirityLabel = new javax.swing.JLabel();
    playerVulnerLabel = new javax.swing.JLabel();
    jLabel13 = new javax.swing.JLabel();
    jLabel14 = new javax.swing.JLabel();
    jLabel15 = new javax.swing.JLabel();
    jLabel16 = new javax.swing.JLabel();
    jLabel17 = new javax.swing.JLabel();
    jLabel18 = new javax.swing.JLabel();
    jScrollPane2 = new javax.swing.JScrollPane();
    jTabbedPane1 = new javax.swing.JTabbedPane();
    jScrollPane1 = new javax.swing.JScrollPane();
    playerRecord = new javax.swing.JTable();
    jTabbedPane4 = new javax.swing.JTabbedPane();
    jTabbedPane5 = new javax.swing.JTabbedPane();
    jTabbedPane6 = new javax.swing.JTabbedPane();
    sendTurnButton = new javax.swing.JButton();
    cancelActionButton = new javax.swing.JButton();
    hideButton = new javax.swing.JButton();
    moveButton = new javax.swing.JButton();
    searchButton = new javax.swing.JButton();
    restButton = new javax.swing.JButton();
    tradeButton = new javax.swing.JButton();
    classLabel = new javax.swing.JLabel();
    setjLabel20(new javax.swing.JLabel());
    jTable2 = new JTable();
    jTable3 = new JTable();
    jScrollPane3 = new javax.swing.JScrollPane();
    jScrollPane4= new javax.swing.JScrollPane();

    //Set the text of components in this view
    getjButton1().setText("Show Card");
    playerLabel.setText("Player:");
    jLabel3.setText("Gold:");
    jLabel4.setText("Fame:");
    jLabel5.setText("Notoriety:");
```

```
jLabel6.setText("Vulnerability:");
getPlayerDisplayLabel().setText("None");
getPlayerGoldLabel().setText("1");
playerFameLabel.setText("2");
playerNotirityLabel.setText("3");
playerVulnerLabel.setText("4");
jLabel13.setText("Badges:");
jLabel14.setText("Colors:");
jLabel15.setText("Curses:");
jLabel16.setText("None");
jLabel17.setText("None");
jLabel18.setText("None");

//Table declarations
playerRecord.setModel(new javax.swing.table.DefaultTableModel(
   new Object [][] {

   },
   new String [] {
      "Turn", "Mon", "Day", "Color", "Phases", "Action", "Kills"
   }
) {
   /**
                     *
                     */
                    private static final long serialVersionUID = 7653752904224053856L;
                    boolean[] canEdit = new boolean [] {
      false, false, false, false, false, false, false
   };

   public boolean isCellEditable(int rowIndex, int columnIndex) {
      return canEdit [columnIndex];
   }
});

jTable2.setModel(new javax.swing.table.DefaultTableModel(
      new Object [][] {

      },
      new String [] {
         "Name","Details"
      }
   ) {
      /**
                     *
                     */
                    private static final long serialVersionUID = 7653752904224053856L;
                    boolean[] canEdit = new boolean [] {
         false, false
```

```
        };

        public boolean isCellEditable(int rowIndex, int columnIndex) {
            return canEdit [columnIndex];
        }
    });

jTable3.setModel(new javax.swing.table.DefaultTableModel(
        new Object [][] {

        },
        new String [] {
            "Name","Details","Status"
        }
    ) {
        /**
                         *
                         */
                        private static final long serialVersionUID = 7653752904224053856L;
                        boolean[] canEdit = new boolean [] {
            false, false
        };

        public boolean isCellEditable(int rowIndex, int columnIndex) {
            return canEdit [columnIndex];
        }
    });

jScrollPane1.setViewportView(playerRecord);
jScrollPane3.setViewportView(jTable2);
jScrollPane4.setViewportView(jTable3);

jTabbedPane1.addTab("Record", jScrollPane1);
jTabbedPane1.addTab("Chits", jScrollPane3);
jTabbedPane1.addTab("Inventory", jScrollPane4);
jTabbedPane1.addTab("Spells", jTabbedPane4);
jTabbedPane1.addTab("Discoveries", jTabbedPane5);
jTabbedPane1.addTab("Victory Requirements", jTabbedPane6);

jScrollPane2.setViewportView(jTabbedPane1);


//Action listeners
sendTurnButton.setText("Send Turn");
sendTurnButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        handleSendTurn();
    }
});
```

```
cancelActionButton.setText("Cancel Action");
cancelActionButton.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      System.out.println("Cancel Action Has Been Pressed");
   }
});


hideButton.setText("Hide");
hideButton.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      handleHideButton(evt);
   }
});

moveButton.setText("Move");
moveButton.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      handleMoveButtonPress(evt);
   }
});

searchButton.setText("Search");

// Button Needs To Be Expanded On And So Do Treasure
searchButton.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent arg0) {
                              System.out.println("Search Button Has Been Pressed");
                              handleSearchButtonPressed();
                    }
          });

restButton.setText("Rest");
restButton.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent arg0) {
                              System.out.println("Rest Button Has Been Pressed");
                              parent.getCurrentPlayer().getCurrentClearing().updateConnectedTiles();
                    }
          });

tradeButton.setText("Trade");
tradeButton.addActionListener(new ActionListener() {
                    @Override
                    public void actionPerformed(ActionEvent arg0) {
                              System.out.println("Trade Button Has Been Pressed");
                              parent.getCurrentPlayer().getCurrentClearing().updateConnectedTiles();
```

```
                    startTrading();
                }
            });

        classLabel.setText("Class:");
        getPlayerClassLabel().setText("None");

        //Layout for this view
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)

            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
                        .addComponent(playerLabel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(jLabel3)
                        .addComponent(jLabel4)
                        .addComponent(jLabel5)
                        .addComponent(jLabel6))
                    .addComponent(classLabel))
                .addGap(18, 18, 18)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(playerVulnerLabel)
                        .addComponent(getPlayerGoldLabel())
                        .addComponent(playerFameLabel)
                        .addComponent(playerNotirityLabel))
                    .addGap(267, 267, 267)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(jLabel15)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .addComponent(jLabel18))
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(jLabel14)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                            .addComponent(jLabel17))
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(jLabel13)
                            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                            .addComponent(jLabel16)))
```

```
                    .addGap(0, 0, Short.MAX_VALUE))
                .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(getPlayerDisplayLabel())
                        .addComponent(getPlayerClassLabel()))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(getjButton1())))
            .addContainerGap())
        .addComponent(jScrollPane2)
        .addGroup(layout.createSequentialGroup()
            .addComponent(sendTurnButton)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(cancelActionButton)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(hideButton)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(moveButton)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(searchButton)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(restButton)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(tradeButton)
            .addGap(0, 26, Short.MAX_VALUE))
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(getjButton1())
                .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(classLabel)
                        .addComponent(getPlayerClassLabel()))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(playerLabel)
                        .addComponent(getPlayerDisplayLabel()))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jLabel3)
                        .addComponent(getPlayerGoldLabel()))
```

```
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jLabel4)
                        .addComponent(playerFameLabel))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jLabel5)
                        .addComponent(playerNotirityLabel))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jLabel6)
                        .addComponent(playerVulnerLabel)))
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jLabel13)
                        .addComponent(jLabel16))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jLabel14)
                        .addComponent(jLabel17))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                        .addComponent(jLabel15)
                        .addComponent(jLabel18))))))
            .addGap(121, 121, 121)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
               .addComponent(sendTurnButton)
               .addComponent(cancelActionButton)
               .addComponent(hideButton)
               .addComponent(moveButton)
               .addComponent(searchButton)
               .addComponent(restButton)
               .addComponent(tradeButton))
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 267,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(15, 15, 15))
      );
   }
```

```
        /*---------------------- Update The View With A Player ---------------------- */
    public void update(PlayerBase p) {
        playerDisplayLabel.setText(p.getName());
        playerClassDisplayLabel.setText(p.getPlayerClass().name());
        playerGoldLabel.setText("" + p.getCurrentGold());
        playerFameLabel.setText("" + p.getCurrentFame());
        playerNotirityLabel.setText("" + p.getCurrentNotirity());
        playerVulnerLabel.setText ("" + p.getVulnerability().name());

        displayRecord(p);
    }

    // Builds The Record Panel Table
    private void displayRecord (PlayerBase p) {
        // Initialize The Data
        ArrayList<String> playerLog = p.getRecordLog();
       String[] headers = {"Turn", "Mon", "Day", "Color", "Phases", "Action", "Kills"};
        String[][] data = new String[playerLog.size()][headers.length];

        // Loop Over The Data
        for (int i = 0; i < playerLog.size(); i++) {
                data[i][0] = "" + i;
                data[i][1] = "";
                data[i][2] = "";
                data[i][3] = "";
                data[i][4] = "";
                data[i][5] = playerLog.get(i);
                data[i][6] = "" + p.getKillCount();
        }

        // Get The Table Model Ready
        DefaultTableModel newModel = new DefaultTableModel(data, headers);
        playerRecord.setModel(newModel);
    }

    // Opens The New Search Dialog
    private void search() {
                PlayerBase currPlayer = parent.getCurrentPlayer();
                ArrayList<TreasureModel> treasures =
currPlayer.getCurrentClearing().getTreasuresPlayerFound(currPlayer);

                // Make The Dialog Appear
                TreasureViewDialog lootDialog = new TreasureViewDialog(parent, "Loot", false,
treasures, currPlayer);
                lootDialog.setVisible(true);
        }

        public void startTrading(){
                JFrame tradeFrame = new JFrame();
```

```
        TradeView tradePanel = new TradeView();
        tradeFrame.setSize(495,355);
        tradeFrame.setLocation(((int)tk.getScreenSize().getWidth()/2)
-   300, ((int)tk.getScreenSize().getHeight()/2) - 300);
    tradeFrame.setVisible(true);
    tradeFrame.add(tradePanel);
    }
    /*---------------------- Event Handler Methods ---------------------------- */
    private void handleSearchButtonPressed() {
    PlayerBase currentPlayer = parent.getCurrentPlayer();
                currentPlayer.getCurrentClearing().updateConnectedTiles();

                if (currentPlayer.getAvailableActions() > 0) {
                        search();
                } else {
                JOptionPane.showMessageDialog(this, "You Have No More Actions For This
    Turn");
        }
        }

      private void handleHideButton(java.awt.event.ActionEvent evt) {
        System.out.println("Hide Button Has Been Pressed");
        PlayerBase currentPlayer = parent.getCurrentPlayer();

        if(currentPlayer == null){
                return;
        }

        // Still Want To Update And Clear The Surronding Tiles If Need Be
        currentPlayer.getCurrentClearing().updateConnectedTiles();

        // If The Player Can Do The Action
        if (currentPlayer.getAvailableActions() > 0) {
                currentPlayer.attemptHide();
        } else {
                JOptionPane.showMessageDialog(this, "You Have No More Actions For This
    Turn");
        }
      }

      // Handles The Move Button Press
      private void handleMoveButtonPress(java.awt.event.ActionEvent evt) {

        System.out.println("Move Has Been Pressed");
        PlayerBase currentPlayer = parent.getCurrentPlayer();

        // If There Is No Current Player Do Nothing
        if(currentPlayer==null){
                return;
```

```
        }

        // If The Player Still Has Actions Available
        if (currentPlayer.getAvailableActions() > 0) {
                currentPlayer.getCurrentClearing().highlightConnectedClearings();
                currentPlayer.setMoving(true);
        } else {
                JOptionPane.showMessageDialog(this, "You Have No More Actions For This
Turn");
        }
    }

    //method used for sending a turn to the controller
    private void handleSendTurn () {
        System.out.println("Send Turn Has Been Pressed");

        // If The Game Has Started
        if (parent.hasGameStarted()) {
                if(parent.getCurrentPlayer()==null){
                        return;
                }

                PlayerBase currentPlayer = parent.getCurrentPlayer();
                currentPlayer.getCurrentClearing().updateConnectedTiles();
                currentPlayer.endPlayerTurn();
                parent.getGameController().moveToNextPlayer();
                parent.getPlayerList().updateTable();
                parent.updateRecordTable();

                // If There Is Multiple Players In The Clearing Then Fight
                ArrayList<PlayerBase> playersInClearing =
currentPlayer.getCurrentClearing().getPlayersInClearing();
                if (playersInClearing.size() > 1) {
                        startCombat(playersInClearing);
                }
        }
    }

        // Shows The Combat In It's Own View
        public void startCombat(ArrayList <PlayerBase> combatingPlayers){
                new CombatView(combatingPlayers);
        }

    /*----------------- Getters And Setters ------------------- */

    public javax.swing.JLabel getPlayerClassLabel() {
                return playerClassDisplayLabel;
        }
```

```
	public void setjLabel20(javax.swing.JLabel jLabel20) {
		this.playerClassDisplayLabel = jLabel20;
	}


	public javax.swing.JLabel getPlayerDisplayLabel() {
		return playerDisplayLabel;
	}

	public void setjLabel8(javax.swing.JLabel jLabel8) {
		this.playerDisplayLabel = jLabel8;
	}


	public javax.swing.JLabel getPlayerGoldLabel() {
		return playerGoldLabel;
	}

	public void setPlayerGoldLabel(javax.swing.JLabel jLabel9) {
		this.playerGoldLabel = jLabel9;
	}

	public javax.swing.JButton getjButton1() {
		return jButton1;
	}

	public void setjButton1(javax.swing.JButton jButton1) {
		this.jButton1 = jButton1;
	}

	public void addToInventory(String theName, String theDetails){
		((DefaultTableModel) getjTable3().getModel()).addRow(new Object[]
{theName,theDetails,"Active"});
	}

	public void addToChits(String theName, String theDetails){
		((DefaultTableModel) getjTable2().getModel()).addRow(new Object[]
{theName,theDetails});
	}

	public void removeFromInvenotry(){
		if(getjTable3().getSelectedRow()==-1){
			return;
		}
		((DefaultTableModel)
getjTable3().getModel()).removeRow(getjTable3().getSelectedRow());
	}

	public void addToRecord(String theTurn, String theMon, String theDay, String theColor,
```

```
String thePhases, String theAction, String theKills){
		((DefaultTableModel) getjTable1().getModel()).addRow(new Object[]
{theTurn,theMon,theDay,theColor,thePhases,theAction,theKills});
	}

	private JTable getjTable1() {
		return jTable2;
	}

	public void removeFromRecord(){
		if(getjTable1().getSelectedRow()==-1){
			return;
		}
		((DefaultTableModel)
getjTable1().getModel()).removeRow(getjTable1().getSelectedRow());
	}

	public void removeFromChits(){
		if(getjTable2().getSelectedRow()==-1){
			return;
		}
		((DefaultTableModel)
getjTable2().getModel()).removeRow(getjTable2().getSelectedRow());
	}

	public void changeItemStatus(String theStatus){
		((DefaultTableModel) getjTable3().getModel()).setValueAt(theStatus,
getjTable2().getSelectedRow(), 2);
	}

	private JTable getjTable3() {
		return jTable3;
	}

	private JTable getjTable2() {
		return jTable2;
	}

	public void massSetButtonState (boolean state) {
		sendTurnButton.setEnabled(state);
		cancelActionButton.setEnabled(state);
		hideButton.setEnabled(state);
		moveButton.setEnabled(state);
		searchButton.setEnabled(state);
		restButton.setEnabled(state);
		tradeButton.setEnabled(state);
	}
}
```

Class: PlayerListView.java

```
package views;

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

import javax.swing.JTable;
import javax.swing.table.DefaultTableModel;

import models.characterModels.PlayerBase;
import models.characterModels.playerEnums.CharacterClass;

        public class PlayerListView extends javax.swing.JPanel {

                //Variable declarations
                private javax.swing.JButton addPlayerButton;
            private javax.swing.JButton removePlayerButton;
            private javax.swing.JButton startGameButton;
            private javax.swing.JScrollPane jScrollPane3;
            private javax.swing.JTable jTable2;

            private GameView parent;

                private static final long serialVersionUID = 1L;

                //Constructor for this view
            public PlayerListView(GameView parent) {
                this.parent = parent;
               initComponents();
            }

            //Initializes the components in this view
            private void initComponents() {

               setjButton1(new javax.swing.JButton());
               setjButton2(new javax.swing.JButton());
               startGameButton = new javax.swing.JButton();
               jScrollPane3 = new javax.swing.JScrollPane();
               setjTable2(new javax.swing.JTable());

               getAddPlayerButton().setText("Add Player");
               getjButton2().setText("Remove Player");
               startGameButton.setText("Start Game");

               //Table declaration
               getjTable2().setModel(new javax.swing.table.DefaultTableModel(
                  new Object [][] {

                  },
```

```
                new String [] {
                    "Class", "Player", "Status"
                }
            ) {
                /**
                                 *
                                 */
                                private static final long serialVersionUID = -7438589133658495131L;
                                boolean[] canEdit = new boolean [] {
                    false, false, false
                };

                public boolean isCellEditable(int rowIndex, int columnIndex) {
                    return canEdit [columnIndex];
                }
            });
                        getjTable2().setSelectionMode(1);

                        //Action listeners
                        getjTable2().addMouseListener(new MouseAdapter() {
                @Override
                public void mouseClicked(MouseEvent e) {
                          JTable target = (JTable)e.getSource();
                     int row = target.getSelectedRow();

                     String test = (String) jTable2.getValueAt(row, 1);
                     CharacterClass characterClass =
CharacterClass.valueOf((String)jTable2.getValueAt(row, 0));
                     parent.updatePlayer(test, characterClass);
                }
                        });

            jScrollPane3.setViewportView(getjTable2());

            //Layout for this view
            javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
            this.setLayout(layout);
            layout.setHorizontalGroup(
                layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                    .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, false)
                        .addComponent(getjButton2(), javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                        .addComponent(getAddPlayerButton(),
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
```

```
                        .addComponent(startGameButton, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                    .addComponent(jScrollPane3, javax.swing.GroupLayout.DEFAULT_SIZE, 709,
Short.MAX_VALUE))
            );
            layout.setVerticalGroup(
               layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
               .addGroup(layout.createSequentialGroup()
                  .addComponent(getAddPlayerButton())
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                  .addComponent(getjButton2())
                  .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                  .addComponent(startGameButton)
                  .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
               .addComponent(jScrollPane3, javax.swing.GroupLayout.PREFERRED_SIZE, 0,
Short.MAX_VALUE)
            );

            // Finally Update The Buttons With The Table Value
            update();
        }

        // Sets The Current Player To Active While The Others Are Waiting
        public void updateTable () {
            PlayerBase currentPlayer = parent.getCurrentPlayer();

            for (int i = 0; i < jTable2.getRowCount(); i++) {
                String playerClass = (String)jTable2.getValueAt(i, 0);
                String playerName = (String)jTable2.getValueAt(i, 1);

                // If The Player Is The Current Player Then Set Them To Playing
                if (currentPlayer.getName().equals(playerName) &&
currentPlayer.getPlayerClass().name().equals(playerClass))
                    jTable2.setValueAt("Playing", i, 2);
                else
                    jTable2.setValueAt("Waiting", i, 2);
            }
        }

        // Updates The Buttons Enability
        public void update (){
            startGameButton.setEnabled(jTable2.getRowCount() > 0);
        }

        public javax.swing.JButton getjButton2() {
                return removePlayerButton;
            }
```

```
	public void setjButton2(javax.swing.JButton jButton2) {
		this.removePlayerButton = jButton2;
	}

	public javax.swing.JButton getAddPlayerButton() {
		return addPlayerButton;
	}

	public void setjButton1(javax.swing.JButton jButton1) {
		this.addPlayerButton = jButton1;
	}

	public javax.swing.JButton getStartGameButton() {
		return startGameButton;
	}

	public void setStartGameButton(javax.swing.JButton startGameButton) {
		this.startGameButton = startGameButton;
	}

	public void addPlayer(String pName, String pClass){
		((DefaultTableModel) getjTable2().getModel()).addRow(new Object[]
{pClass,pName,"Waiting"});
		update();
	}

	public void removePlayer(){
		if(getjTable2().getSelectedRow()==-1){
			return;
		}
		((DefaultTableModel)
getjTable2().getModel()).removeRow(getjTable2().getSelectedRow());
	}

	public javax.swing.JTable getjTable2() {
		return jTable2;
	}

	public void setjTable2(javax.swing.JTable jTable2) {
		this.jTable2 = jTable2;
	}
}
```

Class: TradeView.java

```
package views;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
```

```
import javax.swing.table.DefaultTableModel;

/*
 *
 */
public class TradeView extends javax.swing.JPanel {

    /**
         *
         */
        private static final long serialVersionUID = 1L;

    // Variables declaration
    private javax.swing.JButton sellButton;
    private javax.swing.JButton buyButton;
    private javax.swing.JButton cancelButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTable myInventory;
    private javax.swing.JTable theirInventory;
    // End of variables declaration

    //Constructor for this view
    public TradeView() {
        initComponents();
    }

    //Initializes the components
    private void initComponents() {

        jScrollPane1 = new javax.swing.JScrollPane();
        myInventory = new javax.swing.JTable();
        jScrollPane2 = new javax.swing.JScrollPane();
        theirInventory = new javax.swing.JTable();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        sellButton = new javax.swing.JButton();
        buyButton = new javax.swing.JButton();
        cancelButton = new javax.swing.JButton();

        //Table declaration
        myInventory.setModel(new javax.swing.table.DefaultTableModel(
            new Object [][] {

            },
            new String [] {
```

```
        "Name", "Description", "Value"
    }
) {
    /**
                        *
                        */
                        private static final long serialVersionUID = 1L;
                        boolean[] canEdit = new boolean [] {
        false, false, false
    };

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }
});
jScrollPane1.setViewportView(myInventory);

theirInventory.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {

    },
    new String [] {
        "Name", "Description", "Value"
    }
) {
    /**
                        *
                        */
                        private static final long serialVersionUID = 1L;
                        boolean[] canEdit = new boolean [] {
        false, false, false
    };

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }
});

//Action listeners
buyButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        System.out.println("buy pressed");
    }
});
sellButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
        System.out.println("sell pressed");
    }
});
```

```
        cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                System.out.println("cancel pressed");
            }
        });

        jScrollPane2.setViewportView(theirInventory);

        jLabel1.setText("Your Invenotry:");

        jLabel2.setText("Their Inventory:");

        sellButton.setText("Sell");

        buyButton.setText("Buy");

        cancelButton.setText("Cancel");

        //Layout for this view
        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 215,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(sellButton))
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 19,
Short.MAX_VALUE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(cancelButton)
                            .addGap(99, 99, 99)
                            .addComponent(buyButton))
                        .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 215,
javax.swing.GroupLayout.PREFERRED_SIZE))
                    .addContainerGap())
                .addGroup(layout.createSequentialGroup()
                    .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 92,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 92,
```

```
javax.swing.GroupLayout.PREFERRED_SIZE))))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jLabel1, javax.swing.GroupLayout.PREFERRED_SIZE, 17,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel2, javax.swing.GroupLayout.PREFERRED_SIZE, 17,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)
                    .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED_SIZE, 244,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 244,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
                    .addComponent(sellButton)
                    .addComponent(buyButton)
                    .addComponent(cancelButton))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
    }

    //Getter ,setter, adder, remover
        public void addToInventory(String theName, String theDetails, String theValue){
                ((DefaultTableModel) myInventory.getModel()).addRow(new Object[]
{theName,theDetails,theValue});
        }

        public void addToOtherInventory(String theName, String theDetails, String theValue){
                ((DefaultTableModel) theirInventory.getModel()).addRow(new Object[]
{theName,theDetails,theValue});
        }

        public void removeFromInvenotry(){
                if(myInventory.getSelectedRow()==-1){
                        return;
                }
                ((DefaultTableModel)
myInventory.getModel()).removeRow(myInventory.getSelectedRow());
        }
        public void removeFromTheirInvenotry(){
                if(theirInventory.getSelectedRow()==-1){
                        return;
                }
                ((DefaultTableModel)
theirInventory.getModel()).removeRow(theirInventory.getSelectedRow());
```

```
	}
}
```

Class: TreasureViewDialog.java

```
package views;

import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.Insets;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;

import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;

import models.characterModels.PlayerBase;
import models.otherEntities.TreasureModel;

public class TreasureViewDialog extends JDialog{

	//Variable declarations
	private JButton lootButton;
	private JButton searchButton;
	private JButton cancelButton;
	private JList<TreasureModel> treasureList;
	private JScrollPane scrollPane;

	private ArrayList <TreasureModel> treasures;
	private PlayerBase thePlayer;

	protected GridBagLayout layout = new GridBagLayout();
	protected GridBagConstraints layoutConstraints = new GridBagConstraints();

	private static final long serialVersionUID = -749425455469510620L;

	// Dialog That The Player Will See When Looting
	public TreasureViewDialog (JFrame owner, String title, boolean modal,
ArrayList<TreasureModel> treasures, PlayerBase player) {
		super(owner, title, modal);
		this.treasures = treasures;
		this.thePlayer = player;
		initWindow();
		addListeners();
```

```
}

// Initialize The Window
private void initWindow() {
	setSize (300,300);
	setLayout(layout);
	TreasureModel[] temp = new TreasureModel[treasures.size()];
	temp = treasures.toArray(temp);

	treasureList = new JList<>(temp);
	scrollPane = new JScrollPane(treasureList);
	lootButton = new JButton("Loot Treasure");
	cancelButton = new JButton("Cancel");
	searchButton = new JButton ("Search Clearing");

	// Add The Stuff To The View
	addToFrame(scrollPane, 0, 0, 2, 2);
	addToFrame(lootButton, 0, 2, 1, 1);
	addToFrame(searchButton, 1, 2, 1, 1);
	addToFrame(cancelButton, 0, 3, 2, 1);
}

// Add The Listeners
private void addListeners () {
	lootButton.addActionListener(new ActionListener() {
		@Override
		public void actionPerformed(ActionEvent arg0) {
			System.out.println("Loot Button Has Been Pressed");
			handleLootButton();
		}
	});

	cancelButton.addActionListener(new ActionListener() {
		@Override
		public void actionPerformed(ActionEvent e) {
			dispose();
		}
	});

	searchButton.addActionListener(new ActionListener() {
		@Override
		public void actionPerformed(ActionEvent arg0) {
			handleSearchButton();
		}
	});
}


// Handles The Looting
```

```
	private void handleLootButton () {
		if (thePlayer.getAvailableActions() > 0) {
			thePlayer.getCurrentClearing().playerLootClearing(thePlayer);
			refreshTreasureList();
		} else {
		JOptionPane.showMessageDialog(this, "You Have No More Actions For This Turn");
	}
	}

   // Handles The Searching
	private void handleSearchButton() {
		System.out.println("Find test");

		// If There Is Available Actions
		if (thePlayer.getAvailableActions() > 0) {
			ArrayList<TreasureModel> foundTreasures = thePlayer.searchCurrentClearing();
			if (foundTreasures.size() == 0) {
				JOptionPane.showMessageDialog(this, "You Didn't Find Anything In
Your Search");
			} else {
				refreshTreasureList();
				JOptionPane.showMessageDialog(this, "You Found Some Treasure In
The Clearing. You Can Now Loot It");
			}
		} else {
		JOptionPane.showMessageDialog(this, "You Have No More Actions For This Turn");
	}
	}

	private void refreshTreasureList() {
		treasures = thePlayer.getCurrentClearing().getTreasuresPlayerFound(thePlayer);
		TreasureModel[] temp = new TreasureModel[treasures.size()];
		temp = treasures.toArray(temp);
		treasureList.setListData(temp);
	}

	// Add To This Frame
	protected void addToFrame (JComponent theComponent, int x, int y, int gridWidth, int
gridHeight){
		layoutConstraints.gridx = x;
		layoutConstraints.gridy = y;
		layoutConstraints.gridwidth = gridWidth;
		layoutConstraints.gridheight = gridHeight;
		layoutConstraints.fill = GridBagConstraints.BOTH;
		layoutConstraints.insets = new Insets(10, 5, 10, 5);
		layoutConstraints.anchor = GridBagConstraints.NORTHWEST;
		layoutConstraints.weightx = 1.0;
		layoutConstraints.weighty = 1.0;
		layout.setConstraints(theComponent, layoutConstraints);
```

```
            add(theComponent);
        }
    }
```