# Project 4

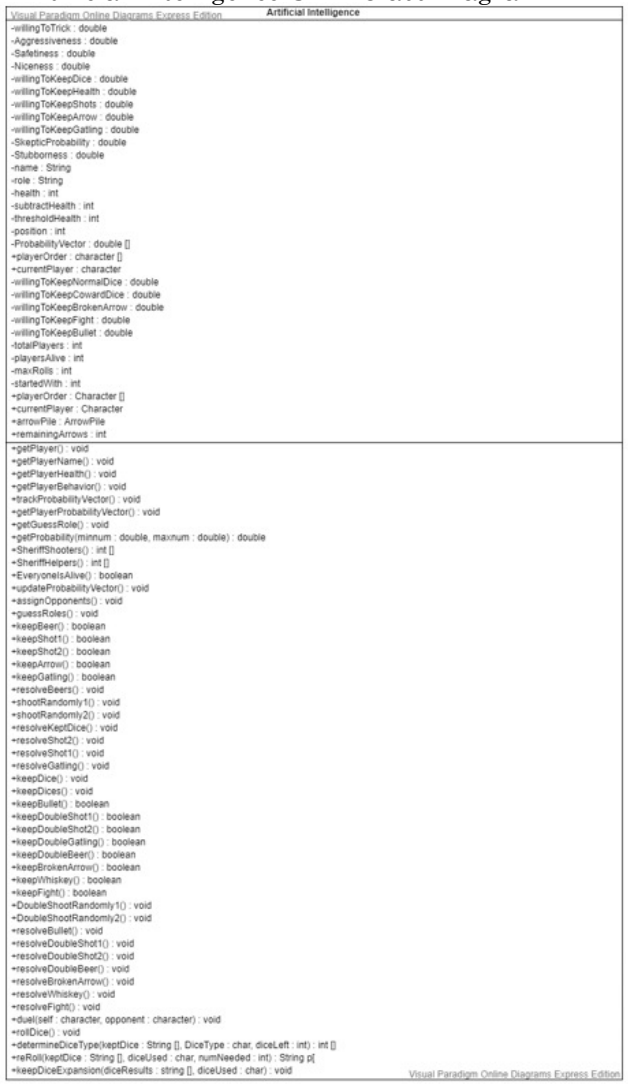
CS2365 Team 4 Presentation

# Team List and Video Presentation

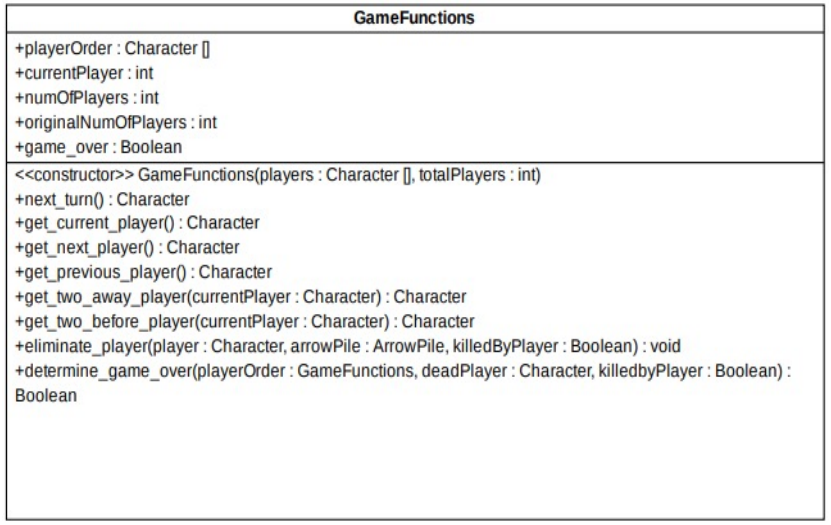
- Team Members:
  - Sadman Ahmed Shanto
  - Cierra Ditmore
  - Meghan Engert
- Video Presentation Link

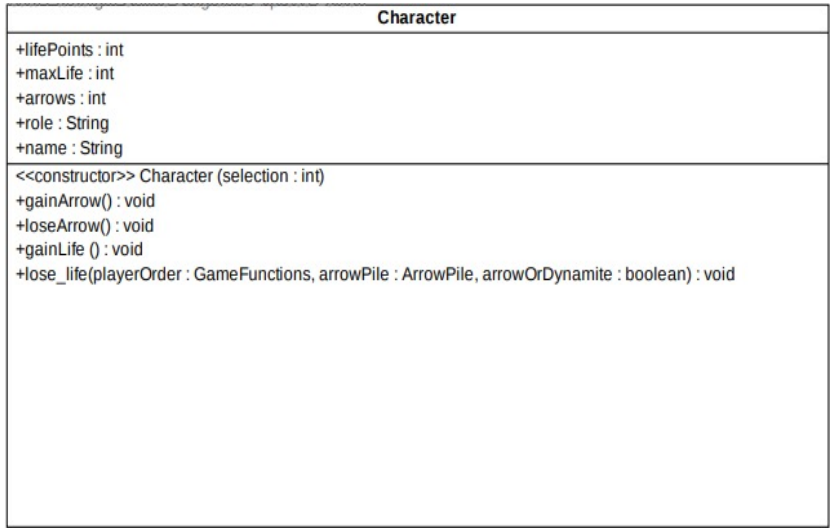
# UML Class Diagrams

Artificial Intelligence UML Class Diagram

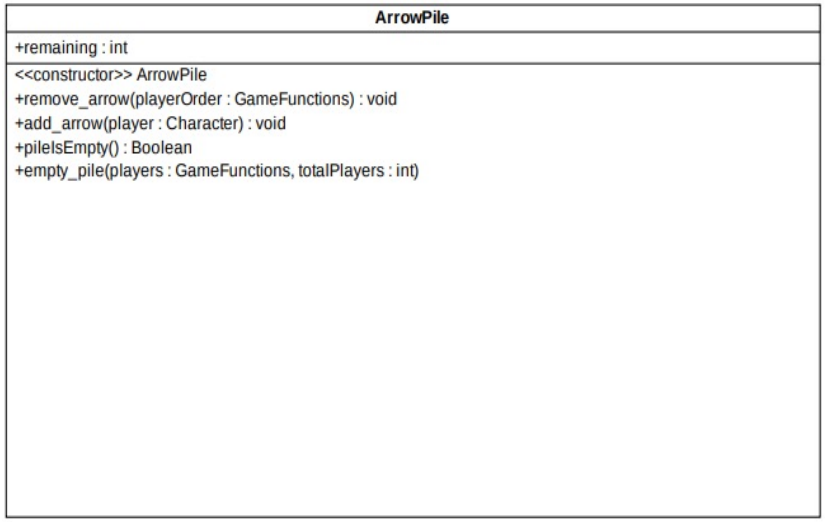


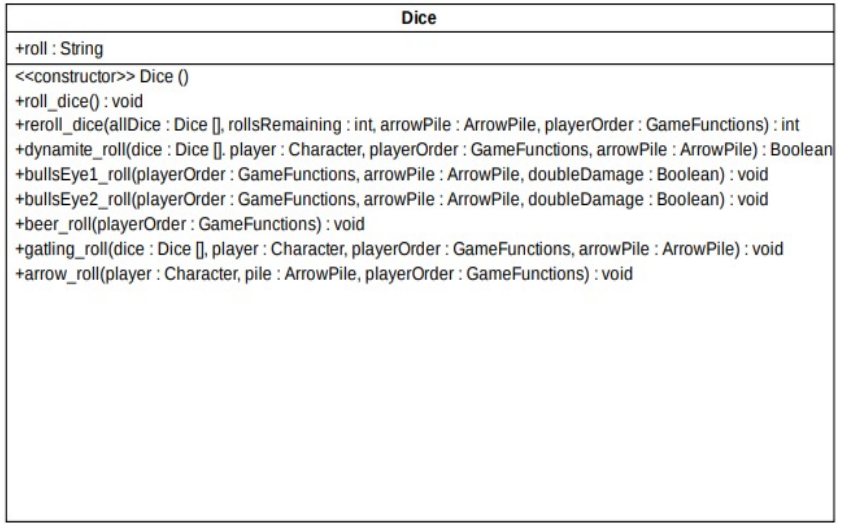

GameFunctions and Character UML Class Diagrams




ArrowPile and Dice UML Class Diagrams



# Classes Developed

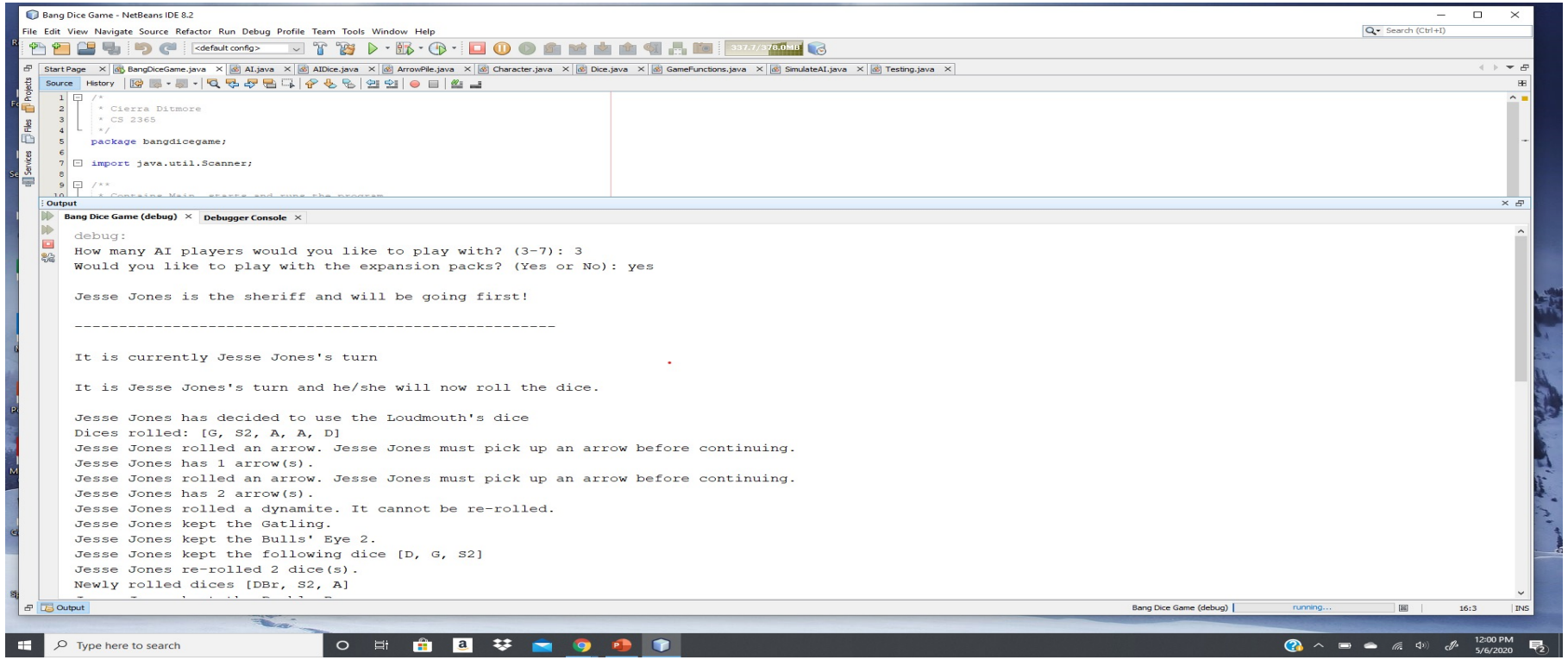
- Game Functions Class
  - Cierra Ditmore
- Dice Class
  - Cierra Ditmore
  - Sadman Ahmed Shanto
- Arrow Pile Class
  - Meghan Engert
- Character Class
  - Cierra Ditmore
- Artificial Intelligence Class
  - Sadman Ahmed Shanto
- Simulate AI Class
  - Sadman Ahmed Shanto
- AI Dice Class
  - Sadman Ahmed Shanto
- Arrow Pile Class
  - Meghan Engert
- BangDiceGame
  - Meghan Engert

# MVC Report: Model

- The Bang Dice Game implemented in Java works correctly with minor bugs in the code.
- All bugs and problems in the code were solved.
- Lack of Team Communication caused this project to lack a GUI because two other members left the group.
- Fixed Bugs:
  - The character class was built using the static arrays, when players died they got a NullPointer exception
  - AI with negative lifePoints were still playing the game.
  - AI could keep more than 5 dice in some cases with more than one dynamite roll.
  - The dynamics of Arrows in the game were not resolved accurately in some test cases.
- Problems Solved:
  - AI:
    - Initially, the program was based on taking user inputs so they were not compatible with the AI, leading to problems with the AI not working simultaneously with a user.
    - Solved by implementing AI classes separately from the initial classes, then making them work with user input.
  - Probability Vectors:
    - Challenging program to build, but critical to the AI
    - Solved by working with code until the probability vectors were solved.

# MVC Report: View

- This project is does not have a GUI due to team difficulties, but the game was fully implemented and runs in the console.

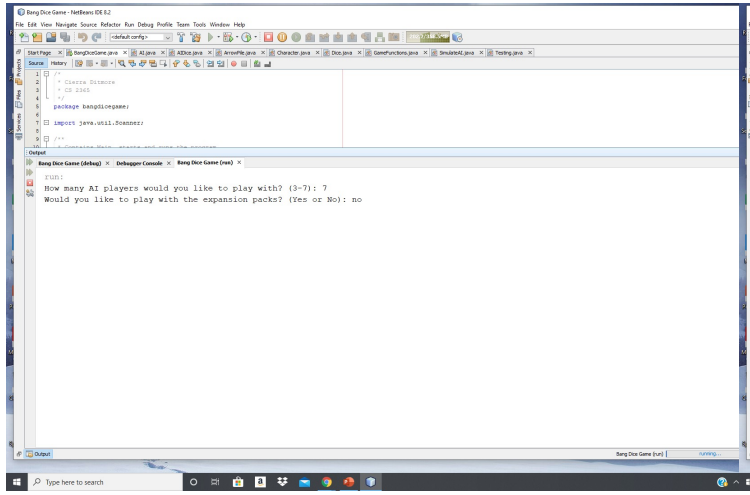



BangDiceGame running in the console

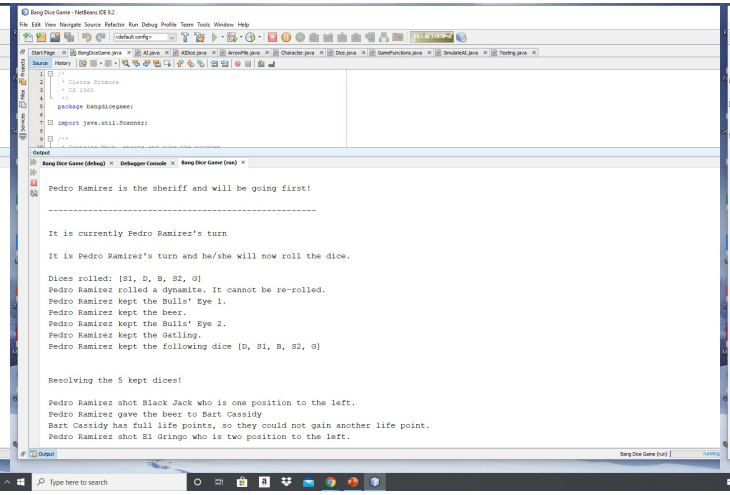
# MVC Report: Controller

- The console is the controller.
- The project is completely integrated with all classes working together so the user can:
  - play the game by giving input to the computer,
  - play with 3 to 7 Artificial Intelligence players,
  - and play the game using a Graphic User Interface (GUI).

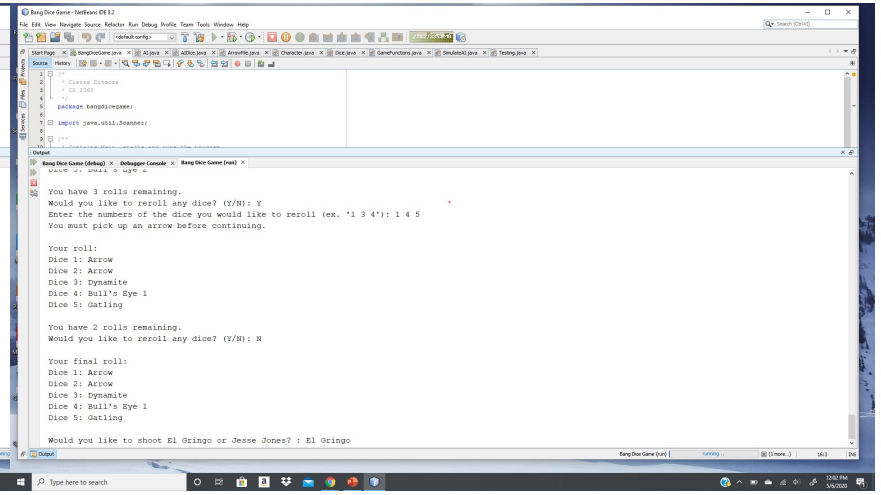
Prompting user for input




AI turn is printed to the screen




Prompting user for their turn



# Code Highlights

- The AI Class could not exist without the SimulateAI Class (composition).
- The AI Dice Class is dependent upon the AI class, thus the class also depends on the Simulate AI class.

```java
public SimulateAI (Character [] players, int totalPlayers, ArrowPile arrowPile){
    this.playerOrder = players;
    this.totalPlayers = totalPlayers;
    this.arrowPile = arrowPile;
    for(int i=0;i<totalPlayers;i++) {
            players[i].ProbabilityVector = createProbabilityVector(totalPlayers);
    }

}

/**
 * runs an AI turn
 *
 * @param game GameFunctions object
 * @param i position in array
 * @param arrowPile arrowPile object
 */
public void AITurn(GameFunctions game, int i, ArrowPile arrowPile) {
        AI aiPlayer = new AI(game, i, arrowPile);
        aiPlayer.turn();
        aiPlayer.rollDice();
        System.out.println();
    }
```

Simulate AI Class

The AI Dice Class

```java
                    while (numRolls!=maxRolls) {

                                                        //determine how many dice needs to be re rolled
                        int diceLeft = diceToRoll-this.keptDice.size();
                        int regular = 0;
                        int saloon = 0;
                        int duel = 0;
                        AIDice d2 = new AIDice();
                        ArrayList<String> newDice =  new ArrayList<String>();//summation of all dices

                        System.out.println(this.currentPlayer.name + " re-rolled " + diceLeft + " dice(s).");
                        newDice.addAll(reRoll(this.keptDice, diceUsed, diceLeft));

                        System.out.println("Newly rolled dices " + newDice);
```

```java
/**
 * initiates a duel between two players
 *
 * @param self current player
 * @param opponent player chosen to duel
 */
public void duel(Character self, Character opponent) {
        AIDice oppoDice = new AIDice();
        //opponent rolls the duel dice
        //if not fight
        if (oppoDice.rollDuelDice() != "F") {
                System.out.println(opponent.name + " did not roll Fight and hence lost the duel.");
                opponent.lose_life(this.game, this.arrowPile, false);
                return; //exit condition
        }
        //opponent.health--
        //if fight
        //self toss
        else if (oppoDice.rollDuelDice() == "F") {
                System.out.println("Since, " + opponent.name + " rolled Fight. It is " + self.name + "'s turn to roll the dice.");
                duel(opponent, self);
        }
    }
```

```java
    public void rollDice() {
        assignOpponents();
        updateProbabilityVector();
        guessRoles();
        int diceToRoll = 5;

        //only rolls 3 dice if a zombie
        if (this.game.get_current_player().role == "Zombie"){
          diceToRoll = 3;
        }
        if (!this.game.expansions) {
                AIDice d = new AIDice();
                //System.out.println(this.currentPlayer.name+ " has decided to not use either of the Coward's or Loudmouth's dice");
                this.diceResults = d.rollThemDice(diceToRoll);
                keepDice(this.diceResults);
                return;
        }
        else {
                AIDice d = new AIDice();
                if (Math.random() <= this.willingToKeepNormalDice) {
                        System.out.println(this.currentPlayer.name+ " has decided to use the regular dies");
                        this.diceResults = d.rollDiceExpansion(diceToRoll, 'R');
                        keepDiceExpansion(this.diceResults, 'R');
                        return;
                }

                else if (Math.random() <= this.willingToKeepCowardDice) {
                        System.out.println(this.currentPlayer.name+ " has decided to use the Coward's dice");
                        this.diceResults = d.rollDiceExpansion(diceToRoll, 'C');
                        keepDiceExpansion(this.diceResults, 'C');
                        return;
                }
                else {
                        System.out.println(this.currentPlayer.name+ " has decided to use the Loudmouth's dice");
                        this.diceResults = d.rollDiceExpansion(diceToRoll, 'L');
                        keepDiceExpansion(this.diceResults, 'L');
                        return;
                }
        }
```

# Code Highlights

- The GameFunctions class makes an array of Character class objects (composition).

GameFunctions Class using Character Class Code Fragment

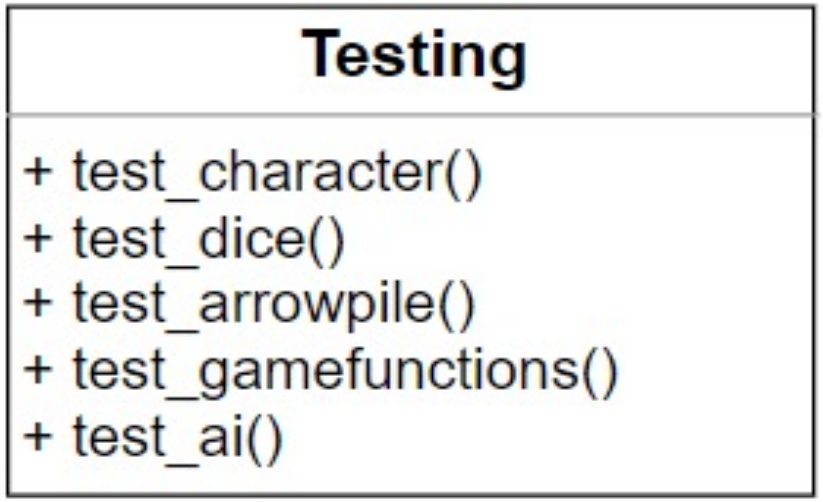
```
public class GameFunctions {

    /**
     * array showing order of player turns
     */
    public Character [] playerOrder;

```

```
    public GameFunctions (Character [] players, int totalPlayers, Boolean expansions){
        int [] deck = {0,0,1,1,1,1,1,1,2,2,2};
        this.playerOrder = players;
        this.currentPlayer = 0;
        this.numOfPlayers = totalPlayers;
        this.numAlivePlayers = totalPlayers;
        this.game_over = false;
        this.realPlayerDead = false;
        this.expansions = expansions;
        this.boneyardDeck = GameFunctions.shuffle_boneyard_deck(deck);
        this.boneyardTotal = 0;
        this.outbreak = false;
    }
```

# Code Highlights

- The Unit Testing for the Character, Dice, ArrowPile, GameFunctions, and AI classes was completed by Cierra Ditmore.

Unit Testing UML Class Diagram



# Project Demo

- Bang Dice Game Video Demo Link

# Submission Files

- UML Class Diagrams, UML Use-Case Diagrams
- Class JavaDoc documentation
- Sourcecode
- Powerpoint file