**CS 210 Software Design Document for Pazaak**

**Made by:**  
Adnan Al Joubi  
Prince Maduekwe  
Kory Stennett

This document helps to track and describe the design aspects of this project. The purpose of this project is to build a version of Pazaak which is a card game slightly similar to blackjack, originally implemented in a Star Wars role playing game. We hope that this document could allow a developer to understand and begin implementing the features described within.



**(Note: Picture pulled from an etsy page, https://www.etsy.com/listing/972446311/pazaak-card-deck-2-side-decks-included)**

Graphical user interface, diagram

Description automatically generatedGraphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

Graphical user interface

Description automatically generated

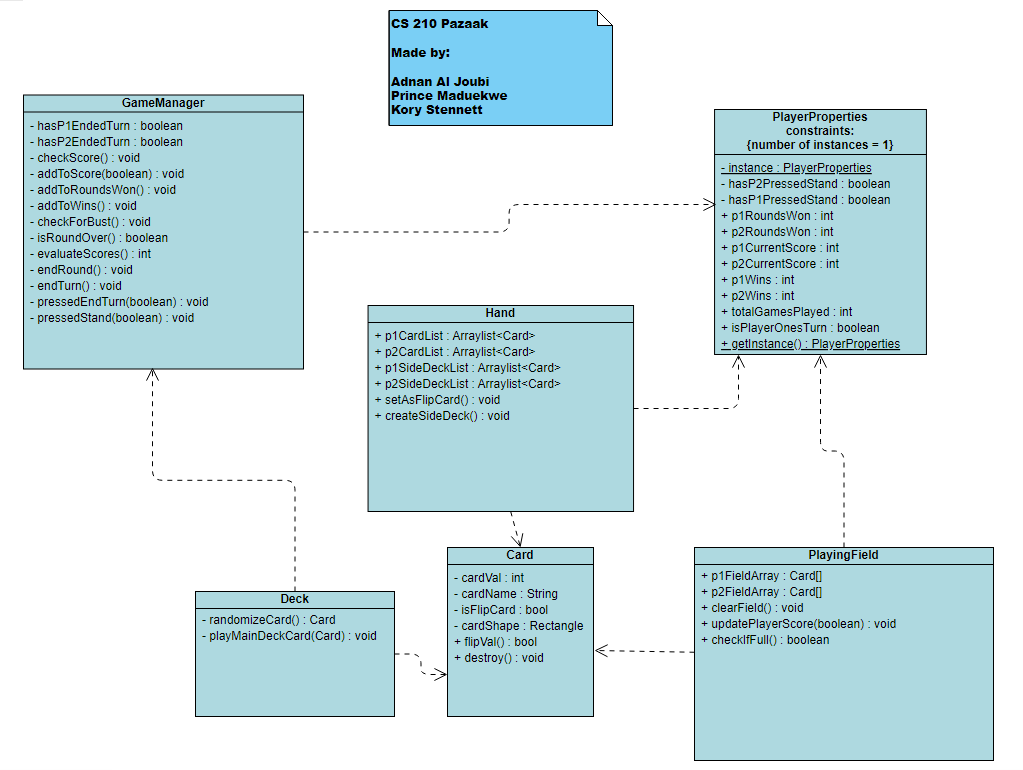
Graphical user interface

Description automatically generated

**Singleton Pattern: PlayerProperties class list of properties**

* boolean hasP1PressedStand
* boolean hasP2PressedStand
* int p1RoundsWon
* int p2RoundsWon
* int p1CurrentScore
* int p2CurrentScore
* int p1Wins
* int p2Wins
* int totalGamesPlayed
* boolean isPlayerOnesTurn

**UML Diagram**

****

**MISC. Extra Information**

**Zoom Notes**

**Need 2 Mockups with detailed info on what happens during a turn:**

**Mockup 1: ARGON**

**Mockup 2: ARGON**

**Need a list of stats that will be tracked in Singleton Class:**

Probably PlayerStats class, tracking p1RoundsWon, p2RoundsWon, p1Score, p2Score.

**Need UML diagram:**

For each class, be sure to include the class name, all instance variables with symbols to denote public/private, and all methods beyond the constructor and get/set methods (while you can include constructors and get/set methods in the UML diagram, they are often omitted to add greater emphasis for the more interesting methods that you will create).

**Which classes are involved:**

* **ARGON Deck** – Creates a random card that will be drawn by each player at the beginning of their turn

**Card variable**.

**RandomizeCard() -> Create random card, always positive, 1-10.**

* **ARGON Card** – gives power over the appearance of the card based on whether it’s positive/negative, and sets the cards value based on user actions, also holds on to its own value
  + **Int Value, Text text, shape/rectangle, bool isFlipCard**
  + **FlipValue() -> switches the sign of hand cards**
  + **Destroy() -> gets rid of the Card, both UI and logic.**
* **PRINCE Hand** – holds a list/array of cards, some cards can be flip cards, some can’t
  + **p1CardList, p2CardList, p1SideDeckList, p2SideDeckList**
  + **SetAsFlipCard() -> sets the drawn hand card to a flip card**
  + **CreateSideDecks() -> creates the decks the hand draws from**
* **PRINCE PlayingField** – possibly holds two arrays of size 9, one for each player, where their played/drawn cards go
  + **P1FieldArray, p2FieldArray**
  + **ClearField() -> Calls on Destroy() method Card for each card in the field at the end of Round**
  + **UpdatePlayerScore(player index) -> add to PlayerStats.playerScore**
  + **CheckIfFull() -> Player wins if their array is full, return Boolean.**
* **KORY PlayerStats** – Singleton, holds player data to be used anywhere that it’s needed
  + p1RoundsWon, p2RoundsWon, p1Score, p2Score, p1Wins, p2Wins, totalGamesPlayed, Boolean isPlayerOneTurn.
* **KORY GameManager** – knows when the game should end, knows the rules
  + **Boolean hasP1PressedStand, hasP2PressedStand, hasP1Ended, hasP2Ended**
  + **CheckScore() -> Checks to see who won round based on player scores once both players have stood**
  + **AddToScore(boolean whosTurn) -> adds to players score based on whos turn it is**
  + **AddToRoundsWon() -> adds to roundsWon, determined by EvaluateScores()**
  + **AddToWins() -> add to player’s win count based on whoever has 3 round wins in PlayerStats**
  + **CheckForBust() -> makes sure current player hasn’t busted after they clicked end turn or stand**
  + **IsRoundOver() -> returns boolean, runs every time a player clicks Stand or End turn, or play from Hand, also checks booleans if player1 or player2 has stood**.
  + **EvaluateScores() -> determine if the round is a draw, or who won based on score, return 0 If no one wins, return 1 if player 1 wins, return 2 if player 2 wins, if either player is over 20, then they lose. This method essentially runs CheckScores() and CheckForBust()**
  + **EndRound() -> calls on EvaluateScores(),**

**If 0, game continues, else, EndGame().**

* + **EndTurn() -> ends turn when players chooses to End Turn access PlayerStats.isPlayerOneTurn and changing the value, then calls on CheckScore()**
  + **PressedEndTurn(Boolean whosTurn) -> determines who pressed End Turn based on who’s turn it is and changes its corresponding Boolean value.**
  + **PressedStand(Boolean whosTurn) -> changes the value for p1/p2HasStood**

Rounds – up to 3 wins for each Player

Turns – the individual turns taken inside of each round