PROJECT PROPOSAL

Casino game - Craps

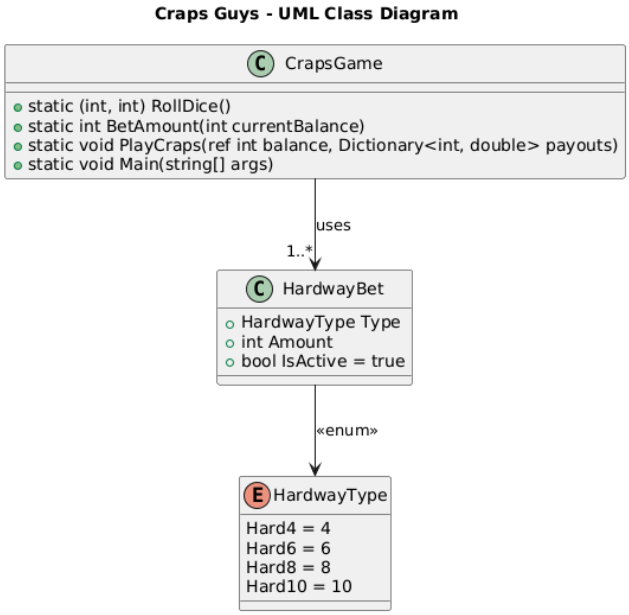
Table of Contents

Project Abstract

This project aims to develop a digital craps/crappless craps casino game that delivers an immersive and interactive gaming experience. The game will simulate real-world craps mechanics, including dice rolls, betting strategies, and payout systems, while incorporating engaging visuals and a user-friendly interface.

Conceptual Design

The Digital Craps Casino Game is a virtual implementation of the classic casino dice game, designed for an engaging and realistic user experience. Any modern device (PC, Mac, or mobile) with an internet connection, a standard CPU/GPU, and at least 4GB RAM for smooth gameplay. JavaScript (for web-based), C# (for Unity), or C++ (for Unreal Engine) or maybe Python (For Pygames) **Unity Engine (C#)** – If 3D animations and physics-based dice rolling are required. **Matter.js or Box2D (JavaScript)** – For realistic physics-based dice rolling.



This UML class diagram represents the structure of our console-based Craps Game application. It's designed to help new developers or teammates understand how the game is organized and how its parts interact. Here are the core functions maintaining the integrity of our application:

CrapsGame (Main Controller Class)

This is the central class that contains all the logic and flow for running the game. All the methods inside it are marked static because this is a utility-style game that doesn't use object instances to run.

- **RollDice()**
  This method returns a tuple of two integers representing the values of two six-sided dice.

- **BetAmount(int currentBalance)**
  A placeholder function that prompts the player to input a bet amount, with validation against their current balance.

- **PlayCraps(ref int balance, Dictionary<int, double> payouts)**
  This is the core function of the game. It handles rolling the dice, resolving standard bets, and processing optional side bets (Hardway bets). It also tracks the player's balance.

- **Main(string[] args)**
  The starting point of the program. It sets the player's starting balance, defines payout rules, and runs the main game loop.

HardwayBet (Side Bet Model):

This class models a **side bet** in the game called a **Hardway Bet**. Hardway bets are wagers that a specific "double" (like 2+2 or 4+4) will appear before a 7 or any other way of making that same total.

- **Type**
  An enum representing which Hardway bet (Hard 4, 6, 8, or 10).

- **Amount**
  The amount of money the player has bet on this particular hardway.

- **IsActive**
  A boolean indicating whether this bet is still in play.

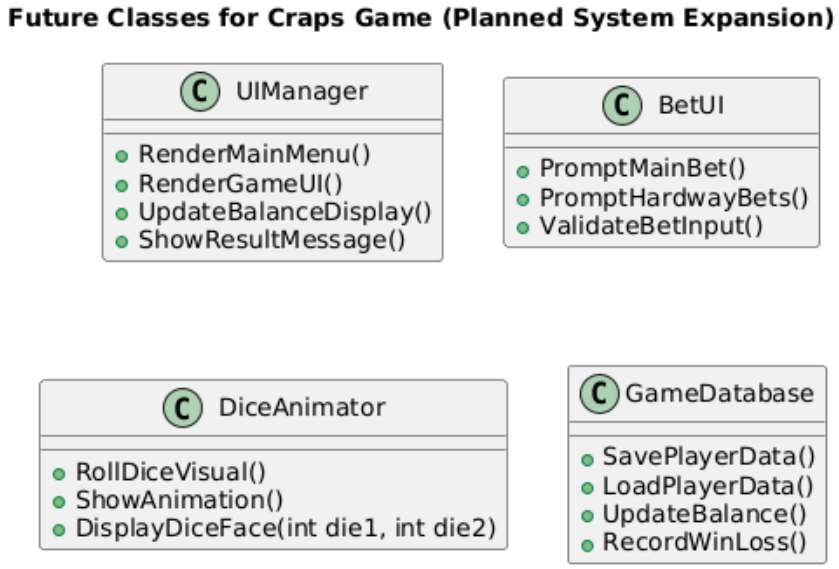
These bets are created inside the PlayCraps method and stored in a List<HardwayBet>.

HardwayType (Enum):

This enumeration lists the valid types of Hardway bets a player can place:

- Hard 4 (2+2)
- Hard 6 (3+3)
- Hard 8 (4+4)
- Hard 10 (5+5)

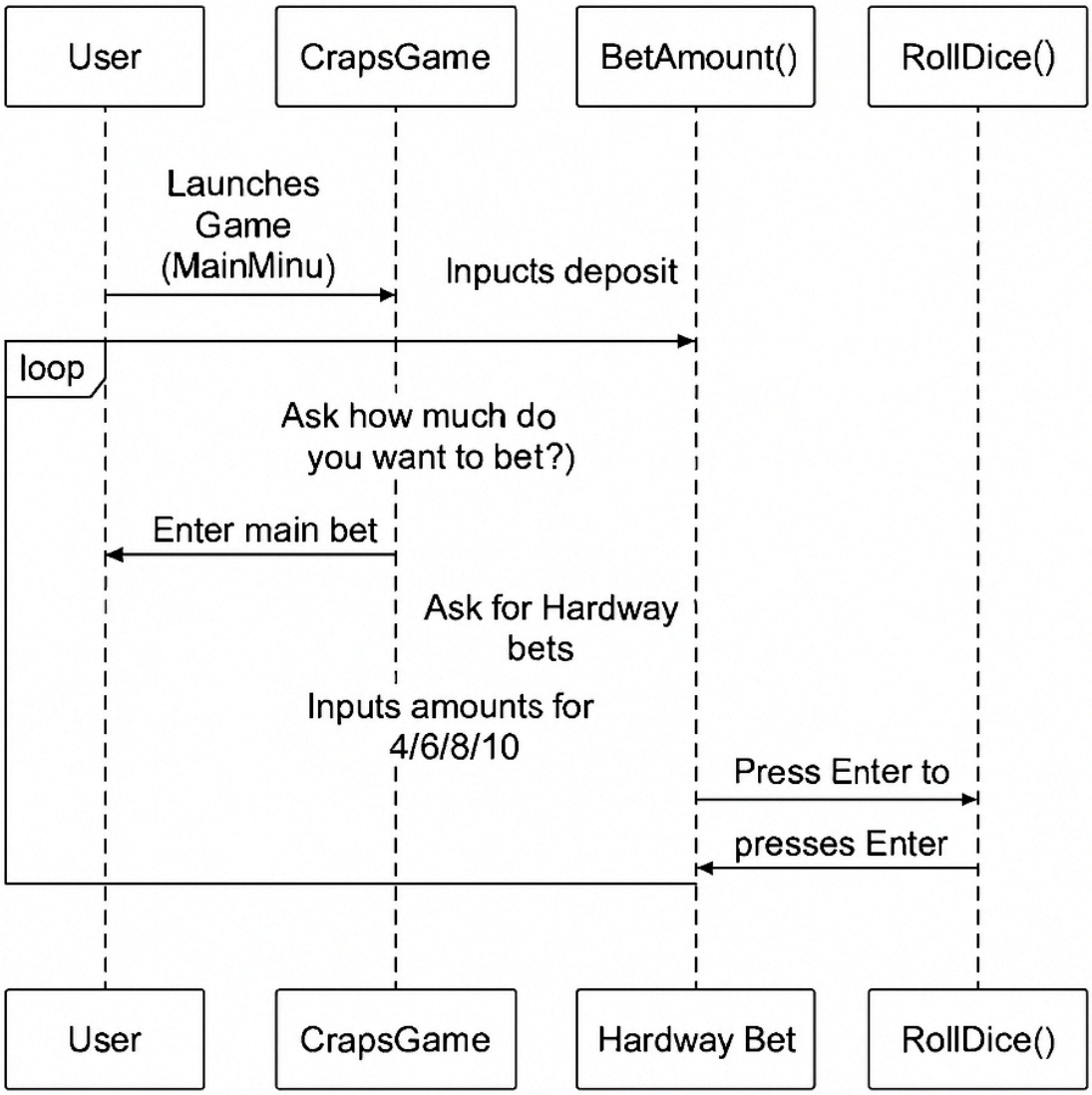
The enum is used by HardwayBet to label the kind of hardway bet and also to display betting options to the user in the console.



The UML diagram above represents future planned components of the Craps Game project. These are not yet implemented but are required based on the project's feature roadmap. These classes are subject to change based on the outcomes of the debugging process, implementation complexity, and performance considerations such as object management and memory efficiency.

- UIManager will serve as the main class responsible for rendering and managing all UI elements in a graphical version of the game.
- BetUI will allow players to input and interact with different betting options via buttons, sliders, or chip selections.
- DiceAnimator will replace the current console-based dice rolls with animated visuals to simulate real dice motion.

- GameDatabase will introduce persistent data storage, allowing the system to track players' sessions, store balances, and manage historical stats across plays.



1. User Action: Start the Game

- The user begins the game by launching the application (via Unity menu or console).
- The CrapsGame class is instantiated and asks the user how much money they want to deposit.

2. Inputting Deposit Amount

- The user inputs a number (e.g., $1000), which sets their starting balance.
- If the amount is invalid, the program sets a default of $1000.

3. Main Bet Placement

- The user is asked to place a main bet using the BetAmount() method.
- This method checks that the bet is valid (not more than current balance, not negative).
- Once validated, the main bet is deducted from the balance.

4. Optional Hardway Bets

- The user is prompted to place side bets on specific Hardway values (4, 6, 8, or 10).
- Each valid side bet creates a new HardwayBet object and deducts the bet from the balance.

5. Dice Roll

- Once all bets are placed, the user presses Enter to roll the dice.
- The RollDice() method is called, simulating two six-sided dice and returning their values.
- The system displays the result and then determines the outcome (win, loss, or point set).

These components lay the foundation for transforming the project from a console-based game to a fully interactive and persistent application.

Proof of Concept

https://github.com/cis3296s25/projects-03-crapsguys/blob/main/CrapsGame/Program.cs - provides a basic text driven implementation of craps, no visual or animations yet.

Background

The proposed digital Craps Casino Game is a software-based simulation of the classic casino dice game, providing players with a fair, engaging, and interactive experience. It aims to capture the excitement of real-world craps tables while enhancing accessibility through an online or downloadable platform. The game will feature realistic dice physics, multiplayer options, AI opponents, statistical tracking, and an intuitive betting system.

Several open-source craps game implementations exist, primarily as simple text-based or web-based simulations. Some notable examples include:

- **"Craps.js" (GitHub Repository)**
  - A JavaScript-based browser craps simulator that focuses on basic game logic without advanced graphics or multiplayer features.
  - **Potential Use:** The proposed game could utilize similar game logic but will be expanded with graphical interfaces, multiplayer capabilities, and an enhanced betting system.
- **"Python Craps Simulator" (Open-Source Project)**
  - A terminal-based craps simulation using Python, focusing on mathematical probabilities and game flow.
  - **Potential Use:** Parts of the probability calculations or logic flow could be referenced, but the proposed game will be built from scratch with more extensive UI and networking features.

This project seeks to **differentiate itself from both basic open-source implementations and commercial gambling-focused craps games** by offering a balance between realism, fair play, educational value, and entertainment. The goal is to **create a high-quality, non-exploitative, and engaging craps experience** for both casual and experienced players. Also there is also a version of craps called crapless Craps which is essentially the same thing with multiple minor differences, which will also be implemented in my version.

Required Resources

- **Craps Rules & Strategies:** Studying the official rules, bet types, payout structures, and player strategies.
- **Probability & Randomization:** Ensuring fair dice rolls using random number generation (RNG) methods.
- **Game Development Principles:** Understanding UI/UX best practices, game physics (for dice rolls), and multiplayer networking.
- **Security & Fair Play:** Researching methods to prevent cheating, such as cryptographic RNG and server-side validation.

- **Comparison to Existing Games:** Analyzing how commercial craps games handle user experience, engagement, and monetization.

Standard PCs / Laptops

- **Windows/macOS/Linux** for development.
- **VS Code or PyCharm** for Python-based development.
- **Unity Engine (C#) or Unreal Engine (C++)** for 3D game development.
- **Node.js or Django (Python)** for backend development.
- **Frontend:**
  - JavaScript (React.js or Vue.js for web version).
  - C# (Unity) or C++ (Unreal Engine) for advanced 3D implementations.
- **Backend:**
  - Python (Django/Flask) or Node.js (Express) for API handling and multiplayer.
- **Database:**
  - PostgreSQL or Firebase (for tracking game sessions, stats, and user profiles).

APIs & Libraries

- **Physics Engines:** Matter.js (for dice physics) or Unity's built-in physics.
- **Multiplayer Networking:** WebSockets (Socket.io), Photon Engine (Unity).
- **Random Number Generation (RNG):** Python secrets module, JavaScript crypto.getRandomValues(), or Random.org API for external validation.
- **Security:** SSL/TLS encryption, Firebase Authentication (for user accounts).

This project will primarily rely on standard CS lab resources but may require additional cloud hosting, physics engines, and game development tools. A discussion with the instructor is necessary to determine access to multiplayer servers, mobile testing devices, and potential GPU requirements for advanced 3D simulations.