CPSC 2720
Team Filthy Casual
Card Game Project

Brett Regnier <brett.regnier@uleth.ca>
Daylend de Grasse <daylend.degrasse@uleth.ca>
Brad Lindsay <brad.lindsay@uleth.ca>
Michael Wynnychuk <m.wynnychuk@uleth.ca>

Team Filthy Casual - Project Document

Table of Contents

Rev: 0.02
October 10, 2016

Revision Date	**Changes**	**Contributor**
October 8, 2016	Initial Document Creation	Brad Lindsay
October 10, 2016	Updated contact information for authors. Added card games to the design section.	Brad Lindsay
October 15, 2016	Added the designs to the document.	Brad Lindsay
Oct 16, 2016	Added player class description to the design rationale.	Michael Wynnychuk and Brett Regnier

Introduction

This group project aims to design and implement 5 text-based card games. These five card games are: Crazy Eights, Blackjack, Phase Ten, Go Fish, and War. This application will be written in C++, and use Violet UML Editor to design the class layouts and hierarchy. Documentation will be generated using the Doxygen tools.

Project Management

Team Organization

- Team Lead - Daylend de Grasse
- Design Lead - Brad Lindsay
- Quality Assurance Lead - Brett Regnier
- Documentation Lead - Michael Wynnychuk

Risk Management

I. Loss of member: If a loss of a member would occur, existing workload would be divided among remaining members. If at all possible the team might request a replacement member.

ii. Unproductive team members: If unproductive members become an issue, a group meeting will occur where the problem will be discussed. The group shall attempt to identify if any legitimate reasons are behind the lack of productivity, and provide support if needed. If there are no legitimate reasons for the lack of productivity, a warning will be issued. If the warning is not sufficient to resolve the lack of productivity, a push for dismissal from the group shall occur.

iii. Inexperienced team members: As it's to be recognized that many members of the group come from different technical backgrounds, and have skills in different areas, the group will try to identify and place members in line with their existing talents. If a member is placed in an area where he/she is inexperienced, that member will be expected to put in effort to become more experienced. If other members of the group have the ability to aid this person in gaining experience, it is their responsibility to share that information to help the individual progress.

iv. Illness: If a member of the group becomes ill, it is their responsibility to inform the group of current work progress, and to onboard other members to aid in progressing their work during the down time.

v. Unanticipated life events: Unanticipated life events may occur during the lifespan of the project. Depending on the severity of the event the following may occur:

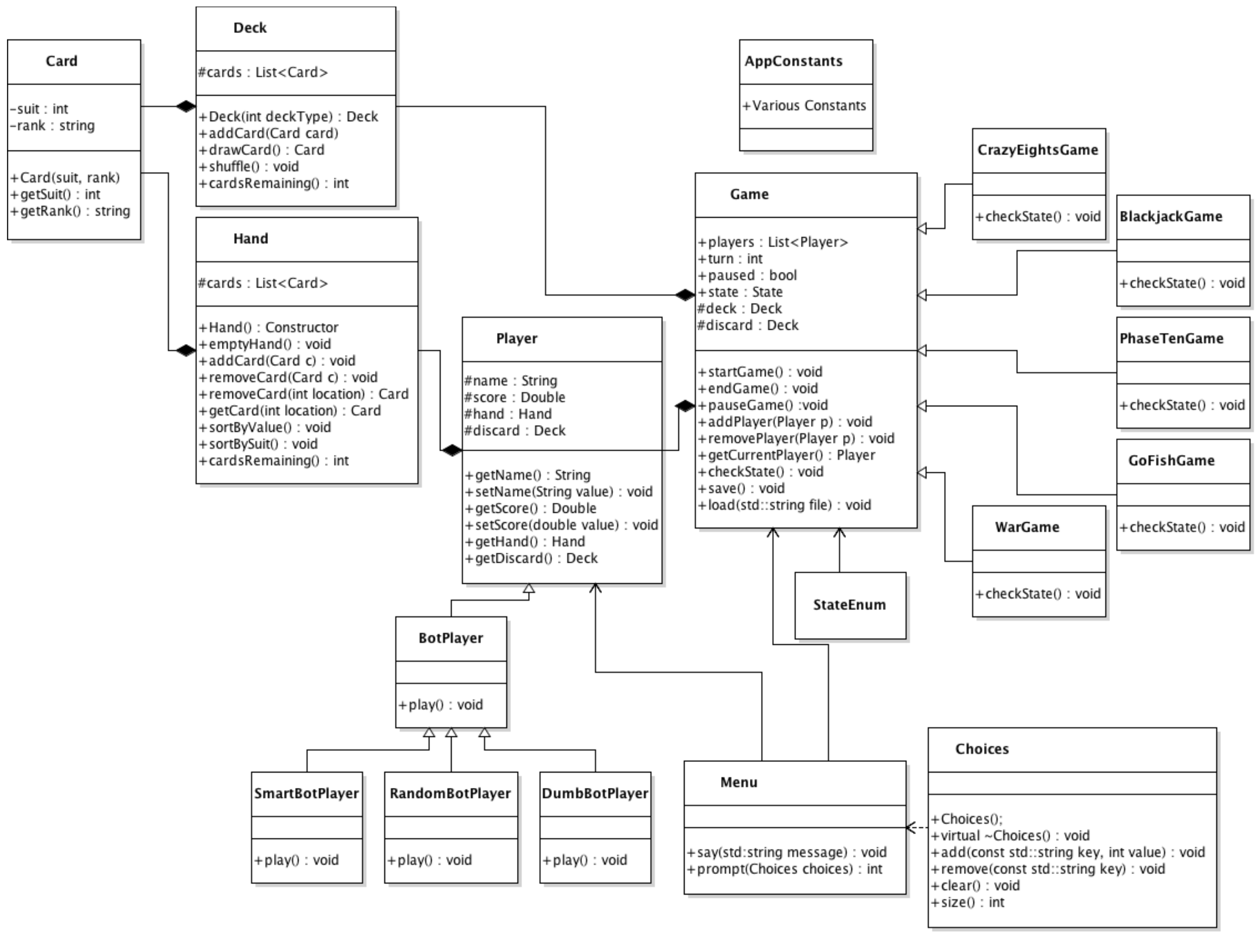
- Recognition of the event
- Delegation of outstanding work
- Communication of work to absent individual
- Reintegration of member into the team if needed

Software Design

The design of this software will most likely change as more discovery is performed. In its current state, the software depends on a simple inheritance chain, and concrete implementations for individual games, and bots. A central module of type Game exists which is a composition of many different facets and will provide the primary API for the user when using the application.

Design

The game has a central module by the name of Game that acts as a base class for all concrete game implementations. This module is a composition of decks, and players. It is also associated with a menuing system, but is not composed of the menuing module. The initial design is described in the UML diagram below:



Design Rationale

This initial design is meant to make the application as extendable as possible without much work needing to be done. There is a central Game class that acts as a primary API for the program. This game class can be instantiated from a game loop inside the main function.

Game acts as a superclass for CrazyEightsGame, BlackjackGame, PhaseTenGame, GoFishGame, and WarGame. These concrete implementations of Game will use the checkState method to enforce game logic. The game class is composed primarily of Players and Decks with methods for managing gameplay. Each game has a Deck which is made up of a collection of cards.

The Player class acts a concrete class for the Human Player along with as a superclass for BotPlayer. The BotPlayer is split apart into 3 concrete classes in which are SmartBotPlayer, RandomBotPlayer and DumbBotPlayer. Each of the BotPlayers will consist of different logic on their Play method according to their type. Each player has a Hand which is made up of a collection of Cards.

The Menu's for the game are associated with both the Player class and the Game class and will provide prompts and messages to the user for required input.

Appendices

Intentionally left blank.