pg7 Design Document – Emily Johnson, Matthias Philippine, Alex Sharata

Overview

Objective:
Two merchants compete head-to-head in a friendly best of three challenge to become the Maharaja's personal trader. The challenge consists of a trading competition where two merchants must collect goods and exchange them for tokens to accumulate points. Bonus tokens are awarded to merchants who make good deals by selling three or more cards at a time. Additionally, the camel token is awarded to the merchant with the most camels in his herd by the end of the round. The merchant with the most points at the end of each round wins a seal of excellence, and the first merchant to win two seals of excellence wins the game.

Materials:
Cards: 10 leather, 8 spice, 6 cloth, 6 silver, 6 gold, 6 diamonds, 11 camels
Tokens: 9 leather, 7 spice, 7 cloth, 5 silver, 5 gold, 5 jewels
Bonus tokens: bonus (6 of each kind), 3 seals of excellence, 1 camel token

Gameplay

Start:
-Tokens in ordered piles based on numbers on tokens, shuffle bonus tokens and make three piles based on cards traded in (3, 4, or 5)
-Market place in middle, put 3 camels and top 2 cards from deck.
-Each player draws 5 cards, camels are revealed, other cards are hidden in hand
-Deck on side in draw pile

Turns:
Take**:**
-Take all camels in marketplace (need to take all) and refill market with drawpile cards, must be 5 cards in marketplace at end of each turn
-Take one resource card and replace with either 1) card from draw pile
-Take multiple cards and replace with card in hand and/or camel
Sell**:**
-Only sell 1 kind of good per turn, have to sell at least 2 cards if of type gold, diamond, or silver.
-Sell cards you want to sell in discard pile, rake as many resource tokens as cards sold, take a take bonus tokens equal to cards sold if applicable: take bonus token according to cards sold (3,4, or 5)

Rounds:
-End if any 3 resource token piles are exhausted
-End if deck is empty and cannot refill market at end of player turn

Camels - Player with most camels at end gets bonus camel token

Winner gets seal of excellence, first with 2 tokens of excellence wins, 3 rounds possible

UI Sketch:

The screen will be split in two major components: the user interface and player hand will occupy the lower half of the screen while the the top half of the screen will belong to the game board.

The game board will consist of a tokens display, marketplace, herd counter, point counter, and seal of excellence counter. The names of the tokens and the number of tokens still available will be displayed in the token display box on the left side of the gameboard. Next to the token display box in the center will be the marketplace where 5 cards will be displayed, and above these will be a deck counter to indicate to players how many cards remain in the deck. To the right of the marketplace, the screen will be split in two to display each player's collection of camel cards, the number of points they have accumulated in the round, and the number of seals of excellence they have obtained. The tokens, marketplace, and counters of each player will be updated as players make their moves throughout the game.

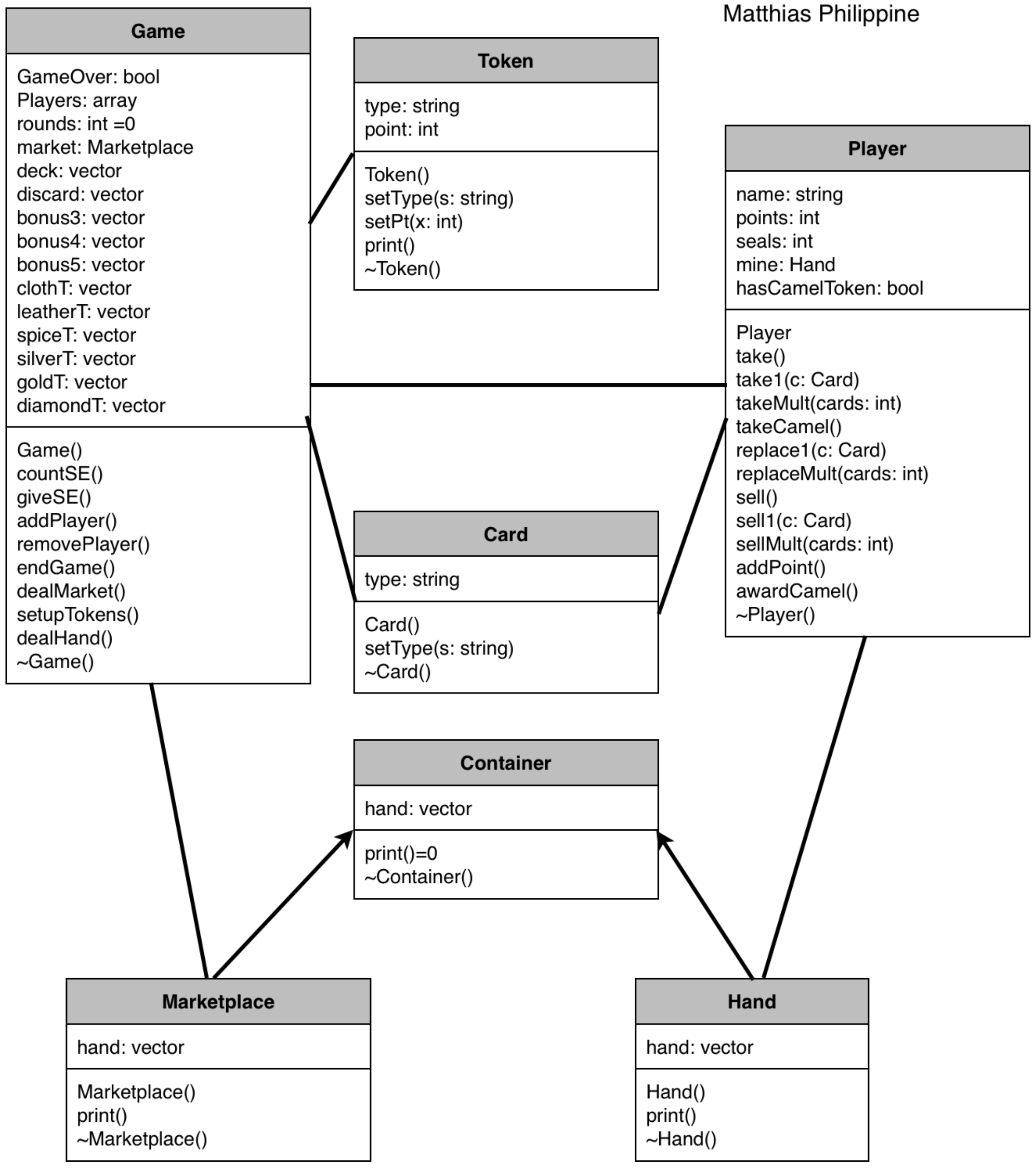
The user interface will house the current player's cards and a command line prompt. The first prompt will ask the player to either 1)Sell or 2)Take. The player chooses which option he would like by inputting the number which appears in front of each option, and will do this for every action he makes.

If a player chooses to Take, then the next prompt will ask which card the player would like to take, giving the player the option to either take all camels if there are any present or take any one card from the marketplace. If the player chooses camels, then the cards he takes will be replaced by those of the deck and the player's turn is over. If the player chooses a resource card, then he is prompted if he would like to take another card or not (y/n input). If no, then the card the player took is replaced by a card from the deck and the player's turn is over. If yes, then the player must choose a card to replace the card he took with a card from his hand. Once the player replaces the card he took, he is prompted again which card he would like to take and which card he would like to exchange until he answers no to the second prompt after choosing a card. Because every turn must be productive, a player cannot exchange a card with the market and then exchange it back to spend his turn having a net zero effect on the game. To apply this rule, every time a player exchanges a card, both the taken card and the replacement card will be unavailable for exchange during the remainder of the player's turn, so the prompts will become less until the player can no longer exchange cards and his turn will automatically end.

If a player chooses to Sell, then the next prompt will ask which type of resource present in the player's hand the player would like to sell. After choosing a type, the player will be prompted how many cards of the chosen type they would like to sell with silk, leather, and spice only needing one card to be sold while gold, silver, and diamonds need at least two cards to be sold. After choosing the number of cards to sell, the player will automatically be awarded a bonus token of equal value to the number of cards sold if there were more than 3 cards sold.

Class Diagram (UML)

Emily Johnson
Alex Sharata
Matthias Philippine



Class Prototypes

Cards.h (below)

```
/*

 Card class
 for all cards (camel, silver, etc)
 holds type of card in string

 */

#ifndef pg7_Cards_h
#define pg7_Cards_h

#include <string>

class Card{
private:
    string type;

public:
    Card(){
        type="Uninitialized Card";
    }

    void setType(string s){
        type=s;
    }

    ~Card();
};

#endif
```

Container.h (below)

```
/*
 ABSTRACT Container Class
 has Cards
 vector for cards
 */

#ifndef pg7_Container_h
#define pg7_Container_h

#include <Cards.h>

class Container{
protected:
    vector<Card> hand;
public:
    virtual ~Container();
    virtual print()=0;
};

#endif
```

Game.h (below)

```
/*
 Game Class
 has players
 has cards
 has tokens
 has marketplace

 to create game, record players, deal game,
 change rounds, determine points and if game is over

 */

#ifndef pg7_Game_h
#define pg7_Game_h

#include "Marketplace.h"
#include "Player.h"
#include <vector>
#define DECKSIZE 55;

using std::vector;

class Game{
private:
   bool GameOver;   //1 if a player has 2 SoE
   Player[2];       //array of players
   int rounds;
   Marketplace market = new Marketplace;

   vector<Card> deck;    //vector deck
   vector<Card> discard;   //discarded cards

   vector<Token> bonus3;   //bonus tokens traded for 3 cards
   vector<Token> bonus4;
   vector<Token> bonus5;

   vector<Token> clothT;   //cloth tokens
   vector<Token> leatherT;
```

```
    vector<Token> spiceT;
    vector<Token> silverT;
    vector<Token> goldT;
    vector<Token> diamondT;


public:

    Game();
    void countSE();  //count Seal of Excellence-->determine game over
    void giveSE();   //give SoE at end of round
    void addPlayer();
    void removePlayer();
    void endGame();      //if game over =1
    void dealMarket();   //set up tokens and cards
    void setupTokens();  //determines order of tokens
    void dealHand();     //gives players 5 cards

    ~Game();

};


#endif
```

Hand.h (below)

```
/*
 Hand Class
 has cards
 creats hand of cards --a vector with max length 7
 */

#ifndef pg7_Hand_h
#define pg7_Hand_h

#include "Container.h"

class Hand : public Container{

protected:
    vector<Card> hand;
public:
    Hand();
    virtual ~Hand();
    virtual print();
};

#endif
```

Marketplace.h (below)

```
/*
 Marketplace Class
 has cards

 holds cards in vector with contant length 5

 */

#ifndef pg7_Marketplace_h
#define pg7_Marketplace_h

#include "Container.h"

class Marketplace: public Container{
protected:
    vector<Card> hand;

public:
    Marketplace();
    virtual print();

    ~Marketplace();
};

#endif
```

Token.h (below)

```
/*

 Token Class
 each token has a string declaring type
 and a point value

 */

#ifndef pg7_Tokens_h
#define pg7_Tokens_h

#include <string>

class Token{
private:
    string type;
    int point;

public:
    Token(){
        type="Uninitialized Token";
        point=0;
    }

    void setType(string s){     //sets type of token
        type=s;
    }
    void setPt(int x){     //sets point value
        point=x;
    }

    void print();

    ~Token();
};

#endif
```

Player.h (below)

```
/*
 Player Class
 has cards
 has hand

 create player, record points, seals, cards
 has all of players actions
 */

#ifndef pg7_Player_h
#define pg7_Player_h
#include "Cards.h"
#include "Hand.h"
#define MAXSIZE 7

class Player{

private:
    string name;    //player name
    int points;     //records points in round
    int seals;      //records #of seals of excellence
    Hand mine= new Hand;      //creates vector from hand class;
    bool hasCamelToken;

public:
    Player()

    void take();
    //functions to take specific items
    void take1(const Card &c);
    void takeMult(int cards);
    void takeCamel();
    void replace1(const Card &c);
    void replaceMult(int cards);

    void sell();
    //functions to sell specific cards
    void sell1(const Card &c);
    void sellMult(int cards);

    void addPoint();
    void awardCamel();

    ~Player();      //need to deallocate hand
};

#endif
```

Implementation Plan

Overall Primary Responsibilities: What the primary responsibilities of each team member will be relative to the overall project (including the design document and demo preparation).

Emily

- Class diagram (including UML)
- .h files
- Core/full functionality

Matthias

- Overview
- UI Sketch
- Demo preparation
- Core/full functionality

Alex

- Managing git repository
- Implementation of UI
- Core/full functionality

Division of Code Labor: Which team member(s) will be primarily responsible for each component of your project code.

Emily

- Finish prototypes for header files
- Filling in core functionality based off prototype (cpp files)

Matthias

- Filling in core functionality based off prototypes
- Creating test files for core functionality (cpp files)

Alex

- Filling in core functionality based off prototype (cpp files)
- Creating user interface for complete gameplay flow

Method of Code Integration: How you plan to integrate work done separately by pairs (recommended) or individuals.

- Utilizing GitHub for our git repository (Master repo: https://github.com/aces9003/pg7)
- We are working closely together as a unit of three, and we will continuously be pushing our individual updates to our git repo so that everyone can pull what the others are working on
    - This way we can each keep all the up to date files of all team members on our respective local machines

- Our approach will allow each of us to maintain a top-down view of the entire project

Development Schedule: The development schedule, which includes deadlines for your various project stages. (Note: integration will take much longer than you expect!)

Due	Part	What
Tuesday 11/18	p7a	Design Document with UI, overview, header files
Tuesday 12/1	**p7b**	Post Thanksgiving break, much of core functionality should be written
Friday 12/5	**p7b**	Project Implementation for core functionality, all classes and functions should be written and connected to have the game start taking shape
Monday 12/15	p7c	Project Implementation with full functionality, includes complete UI and complete gameplay
Thursday 12/18	p7d	Peer Evaluations
finals period	(p7e)	Team Demo - will be coordinated by one driver and two players