**Project Report: An Advanced Aeroplane Chess**

**Introduction**

Aeroplane chess is a modern Chinese chess game based on a kind of British chess. Like other chess game, aeroplane chess have been transplanted to computer platform, and we can find many versions of aeroplane chess game online. However, traditional aeroplane chess seems to lose its attraction among teenagers due to the restricted game mode. In this project, our group created an advanced aeroplane chess game. It can not only perform the function of traditional aeroplane chess, but also add new features to make the game totally different. Additionally, this game supports LAN battle, which means friends can play this game online. We hope by introducing this project, our generation can recall the happiness we had once upon a time.

Here is our group members’ name and contribution list:

(Group members are not changed since the submission of project proposal.)

117010243 Tang Wentian

117010099 Huang Tianjian

117010366 Zeng Lewei

117010289 Wu Runzhong

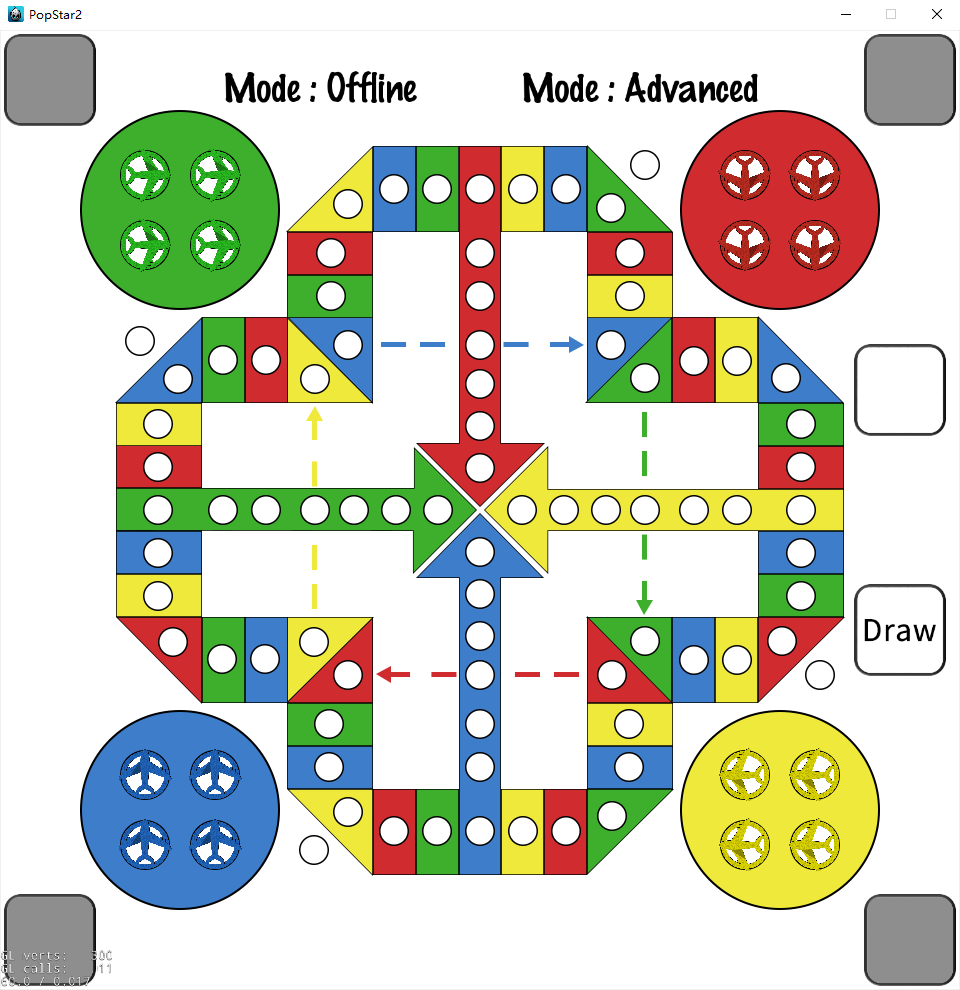
117010155 Lin Siyi

(Job distribution is slightly changed since the submission of project proposal.)

|  |  |
| --- | --- |
| Member | Job |
| Tang Wentian | Scene: start scene, main scene, end scene  Main interface (Normal mode): dice |
| Huang Tianjian | AI design: all parts  Liaison |
| Zeng Lewei | Scene: main scene  Main interface (Normal mode): dice, planes, event dispatch  AI interface: all parts |
| Wu Runzhong | Scene: end scene  Main interface (Normal mode): dice, planes, win judge  Main interface (Advanced mode): all parts |
| Lin Siyi | LAN system: all parts |
| Testing is done by all members.  LAN system is not completed. Other functions have achieved expected results. | |

**Description of the game**

When you enter the beginning interface of this game, you can start a game, read the help, or exit the game. After you start a game, you can change the game mode (online/offline mode, traditional/advanced mode), numbers and type of players.



Game mode:

There are two game modes: traditional mode and advanced mode. Traditional mode is not different from the most common aeroplane chess game. Roll the dice and move the chesses according to the roll point. Let all of your four planes finish the game before other players to win. As for the advanced mode, we add an ability card system to the game, which allow players to use certain ability cards to hinder enemies or buff their own chess to win the game. The mechanism of ability card system is as follows:

1. When a player rolls the dice to 1 or 6, the player gets a chance to draw an ability card instead of moving a chess. One player can only possess at most 1 card at a time. If the player draws a card when having a card already, the original card will be replaced by the new card.
2. One ability card can only be used once, and one player can only use ability card once in one turn. When the player chooses to use ability card, then he or she cannot roll the dice or move chesses anymore.
3. There are four kinds of ability cards, and each of them can be drawn with equal possibility. They are:

* Attack: assign this ability to one OWN chess. Enemy chess less than 5 steps far from this chess will be crashed (send back to airport) unless they are protected.
* Defense: assign this ability to ANY chess. This chess will not be attacked or be interfered (but still can be rammed) in 5 rounds. If a chess has already been interfered, this card will have no effect.
* Interfere: assign this ability to ANY chess. The chess cannot move/attack/defense in 4 rounds.
* Eliminate: assign this ability to ANY chess. The buff/debuff state (defensed/interfered) of the chess will be eliminated.

**Game Framework**

There are three main components in this project, which are main interface (normal mode & advanced mode), AI player (AI core & AI interface) and LAN system.

* **Main interface (Normal mode)**

The basic part of the game.

**Start scene**

**HelloWorldScene**

**Planes**

A new mechanism called EventDispatch should be introduced first. EventDispatch is a mechanism for responding to user events. Event listeners encapsulate your event processing code, event dispatcher notifies listeners of user events, and event objects contain information about the event. Custom event is a kind of event, which is the main method we used in transmitting data between different sprites class. This is because of the running mechanism of Cocos2d-x: All the sprites in the game scene are initialed in the HelloWorld::init() and HelloWorld::setPlane(). To set and use EventDispatch, three steps are needed:

1. Create an EventCustom object, and dispatch it with a unique name.
2. If needed, set user data of custom event into the EventCustom.
3. Set an event listener in the initial function of target sprite, and sign it to the eventDispatcher.
4. The listener will monitor the event. When triggered, it will call the response function.

The structure of using EventDispatch is like a stack. And then the details in planes class can be introduced.

The Planes class inherits the Sprite class in cocos2d.h. Else, it has some data fields to define its color, id, status, buff, position, enter point *(the point of entering outer runway)*, turn point *(the point of entering final runway)*, the start and end point of dotted line, initial direction of planes, coordinate in the airport and take-off point, the plane has jumped or not and the plane can be touched or not.

In the initial function of Planes (init()), all the listeners are signed to eventDispatcher. The create function will call initial function, and set the parameters of the plane.

onTouchBegan is the response function of touchListener. When a plane is clicked, this function will be called. A plane will be click for movement or as a card receiver. For movement, it will first tell other planes that it is being clicked (dispatch event plane\_click). According to the buff and status of plane, different actions will be created and plane will do the action. After each move, the plane will tell dice and other planes that the movement of plane has ended (dispatch event plane\_end), tell other planes the position of itself in order to judge collision happened or not (dispatch event plane\_position), and tell all the planes to summit the status in order to check whether the player has won or not (dispatch event win\_check, further talked in win\_judge part). As a card receiver, this part will be talked in advanced mode part.

When rollPTListener hears the roll number from dice, it will call the response function setRollPoint. This function will set the roll number of planes, helping the plane to create actions and calculate position.

When planeClickListener hears the click event from other planes, it will call the response function setTouchable. It will set the parameter can\_touch of the plane to false, avoiding the plane being clicked during other planes’ action. This is to ensure that only one plane will be click between two roll event.

When roundListener hears the dice roll event, it will call the response function submit\_status. This function will submit the status of plane, in order to help the dice to determine the 4 planes of the player are all unmovable or not.

Function setCard, resetCard, round\_decrease and machinegun\_attack\_judge are designed for advance mode, and will be talked in advanced mode part.

When planePositionListener hears the position of other plane, it will call the reponse function ram\_judge. It will compare the position information of two planes. If they have different color and same position, going\_down function will be called, in order to create crash animation and refresh status of the crashed plane. To set the default texture of crashed plane, set\_texture\_to\_default function will be called.

Function get\_chess, AIMove and AIUseCard are designed for the interface of AI, and will be talked in AI interface part.

**Dice**

**Win judge**

Though win\_judge class inherits the Sprite class in cocos2d.h, it does not act like a sprite. It is added to the scene HelloWorldScene for every game but its position is outside the program window and its size is scaled to 0 (invisible).

For the planes, when the listener winCheckListener hears the event win\_check from a plane with the same color, they will tell the win\_judge whether their status is “finished” (dispatch event finish\_status).

For the win\_judge, listener finishStatusListener hears the event finish\_status and calls function setFinishStatusArray to store the information about whether the status of the planes is “finished”. If all four planes of a player have finished the game, the win\_judge will tell End\_Scene who has won the game and will change the scene to End\_Scene.

**End scene**

* **Main interface (Advanced mode)**

The difference between normal mode and advanced mode is that ability card system is introduced in advanced mode.

**Planes (Advanced mode extension)**

**Dice (Advanced mode extension)**

**Card generator**

The Card\_Generator class inherits the Sprite class in cocos2d.h. It has two data fields: round *(int type, indicating who rolled the dice just now)* and can\_touch *(bool type, indicating whether the player can draw a card)*.

In the initial function of Card\_Generator (init()), all the listeners are signed to eventDispatcher. The create() function will call initial function, and set the parameters of the card generator.

When roundListener hears the player who rolled the dice just now, it will call function setRound to change the parameter round to the integer representing the color of the player.

When rollPTListener hears the roll number from dice, it will call the response function setTouchable. This function checks whether the roll point is 1 or 6. If the roll point is 1 or 6, the parameter can\_touch will be changed to true. Otherwise, can\_touch will be changed to false.

onTouchBegan is the response function of touchListener. The body of the function will be executed if and only if the parameter can\_touch is true. This function generates a random integer representing an ability card (uses function randomInteger extracted from Stanford C++ Libraries) and tells that integer to card slot corresponding to the current player (dispatch event generator\_click). After that, event eventPlaneClick and eventWinCheck are dispatched, since drawing a card consumes a chance to move a chess. After the card generator is clicked, the parameter can\_touch will be set back to false.

Since drawing a card consumes a chance to move a chess, a player cannot draw a card if he or she moves a chess. If planeClickListener hears that a plane is moved (corresponding to event plane\_click), it will call function setTouchableFalse to set the parameter can\_touch to false.

Function AIDraw is designed for the interface of AI, and will be talked in AI interface part.

**Card slot**

The Card\_Slot class inherits the Sprite class in cocos2d.h. It has some data fields: color *(int type, indicating who the card slot belongs to)*, player\_type *(int type, indicating the type of a player)*, card\_num *(int type, indicating the kind of the card in the slot)*, player\_count *(int type, indicating the number of players)*, can\_touch *(bool type, indicating whether the player can use a card)*.

In the initial function of Card\_Slot (init()), all the listeners are signed to eventDispatcher. The create() function will call initial function, and set the parameters of the card slot.

When generatorClickListener hears that the card generator is clicked (event generator\_click) by the owner of the card slot, it will call function setCard to change the parameter card\_num to the integer representing the card generated and to change the texture of the card slot.

When roundListener hears that a new turn begins, it will call the response function setTouchable. This function checks whether it is the turn of the owner of the card slot. If it is, the parameter can\_touch will be changed to true. Otherwise, can\_touch will be changed to false.

onTouchBegan is the response function of touchListener. The body of the function will be executed if and only if the parameter can\_touch is true and the card slot is not empty. This function tells all planes the effect of the card stored in the card slot (dispatch event slot\_click) and then deletes the card in the card slot.

Since using a card consumes an action chance (including rolling the dice), a player cannot use a card if he or she rolls the dice. If rollPTListener hears that the dice is clicked (corresponding to event roll\_point), it will call function setTouchableFalse to set the parameter can\_touch to false.

Function passCard and AIUseCard are designed for the interface of AI, and will be talked in AI interface part.

* **AI core**

The inspiration for this AI core comes from a design of Chinese chess AI. This aeroplane chess AI consists of three basic classes: the move generator (movegenerator.h / movegenerator.cpp), the evaluator (evaluator.h / evaluator.cpp) and the search engine (searchengine.h / searchengine.cpp). Move generator is a class which can generate and store all possible moves according to the current game state. Evaluator is a class which can evaluate the value of a specific move; the larger the value, the better the move. Search engine is the main class for this AI core; it combines initialized move generator and evaluator object in order to simulate a real player’s action. In addition, the AI core contains a define.h in order to store some common data structures and macros for the above three classes. Detailed implementation are as follows:

**define.h**

This head file defines some necessary data structures and macros for the AI. For example, the representation of chess ID, chess colors, chessboard coordinates and etc. They are configured as some macros for the convenience of programming.

There are three important data structures: CHESS, CHESSMOVE, and COORDINATE. CHESS is a structure containing the ID (representing the chess uniquely), the color, the coordinate, buff state and buff round left of a chess. CHESSMOVE is a structure containing the ID of the chess being manipulated, the roll point and the information about ability card using. COORDINATE is a structure containing a string representing the region and an integer representing the location. The reason for using this coordinate representation is that the chessboard of aeroplane chess is not full-covered. Some points on the two-dimensional plane is out of the chessboard, and chess will have different behavior in different region of the chessboard. Therefore, I did not choose two-dimensional coordinate in the convenience of computation.

**Move generator**

When creating a move generator, the constructor will set off mode (deciding what roll point can take off), set the game mode (traditional mode or advanced mode), and set random seed. Setting random seed when initializing will let the dice roll result different every time.

The core function for move generator is createPossibleMove(). This method generate currently possible move according to current chessboard, roll point, the color of the player and ability card the player have. Each possible move is constructed into a CHESSMOVE structure and is stored into the moveCount array.

**Evaluator**

Evaluator only have one function evaluate(). The main logic for evaluating is to compare the chessboard before and after a certain move. If the new chessboard results in a favorable situation, the value will be higher. If the new chessboard confronts a bad situation, the value will be lower.

Because of the mechanism of aeroplane chess, the evaluation process is basically a simulation of human player’s thinking mode. First, the evaluator counts benefits of a move. Benefits include bonus from using ability card efficiently, crashing enemy chess, moving own chess ahead, etc. Then the evaluator counts damages of a move. Damages include the threat from enemies and punishment from overlapping. The final value is counted by benefits subtracting damages.

**Search engine**

This is the main part of AI. When initializing, the search engine must point to a move generator object and an evaluator object in order to function. The play() method will let the AI change chessboard intelligently. This method simulates player behavior like this:

1. Copy the chessboard into cur\_Chessboard which is inside the search engine instance;

2. Roll the dice. Then create move with createPossibleMove().

3. If there is only one move, it must be "not possible to move" case. Then do not do any movement. If there is more than one possible move, search a good move and make move with three methods, searchAGoodMove(), makeMove(), useAbility().

4. If game is not over and roll point is six, continue until the third time roll. Else, player end its turn.

5. Copy the modified chessboard in place to the original one.

searchAGoodMove() will evaluate all possible move and choose the move with the maximum value. makeMove() and useAbility()will modify chessboard according to the move AI choose.

* **AI interface**

This is the interface for game in order to dock to AI core. It coordinates different components of the game, and transmits data between main interface and AI core.

**Before AI interface**

This part starts from the Dice::onTouchBegan. When the dice was being click, it will determine the current player is AI or not. If not, it will do as talked before. Else, it will call Dice::AICall function. The dice becomes untouchable, and it will call all the card slots (eventGetCard) and planes (eventGetChess) in order to provide essential data for AI player. It will also pass

The card slots receive it by getCardListener, and call the response function Card\_Slot::passCard. According to the color of current player, it will pass the corresponding card number to AI interface (eventReceiveCard). And the planes receive it by getChessListener, and call the response function Planes::get\_chess. It will convert the information of itself to the format that AI interface need, and pass to the Dice(eventChessPass). The dice will listen it with chessboardListener, calling response function AIPass to combine all the chesses into a chessboard and pass it to the AI\_interface (event\_receive\_chessboard) with roll point (event\_receive\_point). Now it’s the time for AI\_interface.

**AI interface**

The AI\_interface defines an AI\_player class, which inherits Node class and searchEngine class. First, chessboardListener, pointListener and cardListener will catch the information from dice and card slot with response functions saveChessboard, savePoint and saveCard. After saving them, AIPlay function will use the method in AI core part and calculate the best move. According to the best move created by search engine, there are four situations: don’t move the plane, move the plane, draw card or use card. This matches four different EventCustoms that will be passed: eventAINoMove, eventAIMove, eventAIDraw and eventAIUseCard.

**After AI interface**

Dice will receive the four EventCustoms first, because except for using card, dice need to display the roll animation. Now enumerate the four situations.

* + 1. Don’t move the plane: AINoMoveListener will call response function AINoMove, which only display the animation and set the dice touchable.
    2. Move the plane: AIMoveListener will call response function AIMove. It displays roll animation, passes the target plane’s information to Planes class (eventAIMove2Plane). AIMoveListener in Planes class will call response function AIMove. If the information of plane is match, it will make the planes move.
    3. Draw card: AIDrawListener will call response function AIDraw. It displays roll animation, and call the card generator (eventAIDraw2Generator). AIDrawListener in Card\_Generator class will call response function AIDraw, which is almost the same as the OnTouchBegan function.
    4. Use card: AIUseCardListener will call response function AIUseCard. It will pass the information to card slot (eventAIMove2Slot). AIUseCardListener in Card\_Generator class will call response function AIUseCard, which is almost the same as the OnTouchBegan function.

In Dice class there is an AISkipTurn function, which is similar to SkipTurn function and used for AI using card.

* **LAN system**

**Reference**

Guo Zijian. (2009). Chinese chess online game system.

“random.h” and “random.cpp” from Stanford C++ Libraries. Retrieved from https://stanford.edu/~stepp/cppdoc/

GitHub

Cocos 2d-x documents. Retrieved from https://docs.cocos2d-x.org/cocos2d-x/en/