User Documentation

This is a single player shooting game, named “Saucer”.

To start the game, the player should change directory to game’s directory, then type in “./saucer” in the terminal window.

After the welcome menu shows up, a player could choose to view the high score board by typing H, start the game by typing S, or quit the game by using Q. The instructions are printed when the play types I.

While enjoying a easy shooting level, the player would use “,” and “.” to control the position of the launcher, moving the launcher to the left or right. Once a saucer appears, the player could place the launcher properly and press “SPACE” to shoot a rocket, which could destroy multiple saucers in the same coordinates but never saucers above it.

As soon as the player scores enough points to go to level up, before entering the next level, the player would be asked if he/she would like to trade some scores for extra rockets.

If entering the store, the player could choose 1 of the 3 options or entering “Q” to quit the store and enter the next level up straight.

The game would go on until the player wants to fire a rocket when there is no rocket available. Or the the number of escaped saucers has reached a limit.

For a more humane purpose, “pause and resume” functionality is added to the game. A player could press P for pause during a game and press R to resume the game.

When the game stops, the high score board would be printed out. If the player’s final score is high enough to be recorded into the high score board, then the game would prompt for the player’s username and write the new record.

Developer Documentation

Code Structure

OVERALL STRUCTURE:

highscore.h:

The header file of the highscore.c, which is responsible for printing and editing the high score board.

In highscore.h, a struct highscore is used as the high score board, which has three rows. It’s shown as below:

struct highscore{

```
    char *s1;
    char *p1;
    char *s2;
    char *p2;
    char *s3;
    char *p3;
    int count;
};
```

And also, functions that required by printing the high score board and modifying the high score board are declared in this header file.

highscore.c:

This file is responsible for all behaviours of a high score board: populating the score board with information from a file, adding new score and new username to the board, printing the score board out to the screen.

```
 * highscore.c
 * printStruct()          : print a highscore struct in three lines
 * populate()             : populate a highscore struct with information
 *                          from score.txt file
 * printHighscore()       : print the high score board
 * lowestHighscore()      : return the lowest highscore in current score
 *                          board
 * writeHighscore()       : add new highscore record to high score board
```

game.h:

The header file for the backbone of this saucer game. In game.h, basic functions are declared and the all global mutex and variables and import structures are declared as well.

IMPORTANT DATA STRUCTURES:

Saucer:
Each saucer is a struct type variable which consists of multiple necessary values and is shown as below.

```
struct   saucer {
    char *str;    /* the string represents the appearance of a saucer, which should be <
                     —->*/
    int row;      /* the row indicates the row number of the saucer*/
    int col;      /* the col indicates the column number of the saucer*/
    int hit;      /* the hit is a flag that shows if the current saucer is hit by a rocket*/
    int delay;    /* delay determines the time between each move */
    int live;     /* If a saucer is in spawn process, live is 0, otherwise it's 1*/

};
```

Rocket:

```
struct rocket {
    int speed;   /* speed determins the time between each move*/
    int row;     /* the row indicates the row number of the rocket*/
```

```
        int col;    /* the col indicates the column number of the rocket*/
    };
```

game.c:

The game.c files contains all implementation of declared functions in game.h, and the main function is responsible for running the game.

```
*   Main game:
*     getLimit()             : get the limit of escaped rockets for each level
*     getReward()            : get the reward value for each correct shot in each
*                              level
*     levelUpLimit()         : get the limit of scores for user to level up
*     getDelay()             : return the delay value for a saucer thread
*     getRocketDelay()       : return the delay value for a rocket thread
*     updateSetting()        : update all information like limit, score, rocket in
*                              level up process
*     gameOn()               : the game function that runs in each level
*     setup()                : setup function initializes necessary variables for game
*                              to start
*     moveLeft()             : move launcher to left
*     moveRight()            : move launcher to right
*     printUserMenu()        : print welcome menu
*     printInstruction       : print user instruction
*     levelup()              : modify necessary variables and level up
*     updateStatus()         : update the status line at the bottom of the screen
*     enterShop()            : prompt user to buy rockets by trading scores
*     recordHighscore        : record a user's score if the score is higher than the
*                              lowest score in highscore board
*     unlockEverything       : unlock all locks for game to resume
*     lockEverything         : lock every lock to pause the game
*
*   Rocket:
*     moveRocket()           : move the rocket up.
*     disposeRocket()        : dispose a rocket by eliminating it from screen
*     hitReward()            : when hit a saucer, this function will add proper scores
*                              and rockets to user's current scores and rockets
*     *fire()                : a rocket thread that moves up and detects if it hits a
*                              saucer
*   Saucer:
*     moveSaucer()           : move the saucer from left to right
*     vanish()               : let a saucer escape the screen character by character
*     *attack()              : a saucer thread that moves saucer and tries to escape
*     spawn()                : after got hit or escaped, a saucer thread is recycled
*                              and put to spawn process.
```

FLOW AND FUNCTIONALITIES:

Start-Up:

Starting from the main function, printUserMenu() and printInstruction() would be called first so that the game could give the user a nice user interface which shows what options a player could take, and the instruction about how to play the game. Then the main function will be waiting for the player's input. Depending on the input, if "H" is pressed, the high score board will be printed out and the main function will not stop prompting for the user input; else if "Q" is pressed, the game will be terminated; else the "S" is printed, this will lead to the break out from the loop of prompting for the user input and the first level of the game.

Configuration Before Each Level:

Before starting each level, configuration for the level will be done in setup() function if the level is one, otherwise it would be done in levelup() function. In the configuration phase, depending on the level, each saucer thread will be initialized with corresponding setting; the launcher will be placed in the centre in the second last line of the screen; status information: the number of available rockets, current score, the number of escaped rockets, the level will be displayed in the very last line of the screen. If current level is greater than 1, the game will print out the "level up message", which is composed of an ascii work for "level up" and related information to the next level, for instance, the number of scores to the next level, the limit of escaped saucers; also the game store will become available to user and let the user decide if he/she needs to buy some rockets.

After configuring the screen and status information, saucer threads are created and assigned to *attack(void *arg) function, in which a saucer will appear from the very left of the screen and move towards the right edge of the screen by function moveSaucer(struct saucer *saucer). In order to make the escape of a saucer smoother, I make the saucer disappear character by character, and leaving bigger chance for the player to hit a saucer.

Game:

*Rocket:

In the game play, a rocket thread is created dynamically once the user presses space. And the thread function would be *fire(void *arg), in which, before the rocket escapes the screen, it will move automatically by moveRocket(struct rocket *rocket) function, and when it's in the saucer area, it will check if it hits a saucer. If successfully hitting a saucer, disposeRocket(struct rocket *rocket) will remove the rocket from the screen and hitReward(int countHits) will reward the player with a number of rockets and scores that varies as the number of hits varies.

As soon as the rocket gets destroyed or it escapes the screen, the rocket thread will exit.

*Saucer:

While trying to escaping from the screen, the saucer could be hit by a rocket so that the saucer will disappear by function vanish(struct saucer *info, int count), and get spawned in function spawn(struct saucer *info). Thus it will appear later on.

As soon as the score accumulates enough to enter the next level, the game play on current level will stop, and go to the configuration phase again.

Threads

Main thread:

The main thread is responsible for handling the user input and running the game, including positioning the launcher, controlling gaming in each level, terminating the game when in need, the store mode between each level, and printing the high score board and instructions.

Input thread:

Input thread handles the input from the player during the game play.

Saucer threads:

Each saucer thread has a main function named attack that has multiple sub functions which will take control of the movement of a saucer, and its escape, and the spawn process after either it is hit or has escaped.

Rocket threads:

Each rocket thread will have a function that controls the movement of a rocket, a function that wipe the rocket off the screen once it hits a saucer. Additionally, a function named hitReward is responsible for adding scores and rewarding the player reasonable amount of rockets each time the rocket hits a saucer.

Critical Sections

There are 6 global mutex, which are:

```
/* mutex for modifying window*/
pthread_mutex_t mx = PTHREAD_MUTEX_INITIALIZER;

/* mutex for accessing a saucer's position and hit value */
pthread_mutex_t rk = PTHREAD_MUTEX_INITIALIZER;

/* mutex for modifying escape variable */
pthread_mutex_t es = PTHREAD_MUTEX_INITIALIZER;
```

```
/* mutex for decreasing/ increasing the number of rockets*/
pthread_mutex_t dc = PTHREAD_MUTEX_INITIALIZER;

/* mutex for modifying score*/
pthread_mutex_t sc = PTHREAD_MUTEX_INITIALIZER;

/* mutext for modifying level variable*/
pthread_mutex_t lv = PTHREAD_MUTEX_INITIALIZER;
```

And corresponding global variables:

```
struct saucer saucer[5];  /* a saucer array*/

int escape = 0;             /* the number of escaped saucers*/

int level = 1;              /* the current level*/

int rockets = 10;           /* the number of rockets*/

int score = 0;              /* the score of current player*/
```

Thus any section that modifies the a saucer's values, or tries to increase the score, the escape or the level, or decreases the rockets should definitely be a critical section. Additionally, for display, any section that writes or erases the screen is a critical section.

And more specifically, a rocket thread could modify the variable score and rockets, and have access to the values of the array of saucer since a rocket thread needs to know if it hits a saucer.

A saucer thread needs to modify the variable escape.

Known Bugs:

• Sometimes after entering the username in high score board, the player can't resume to type in terminal normally.
• There could exists some latency in saucer movement rarely. It may look like the rocket is hitting the very edge of a saucer, however, because of the latency, the saucer actually has moved towards left and is not in the hitting range of the rocket, but it only appears to be hit.