COP 290 Assignment 3

Space Invaders

Faran Ahmad
2013CS10220

Kabir Chhabra
2013CS50287

Kartikeya Gupta
2013CS10231

Prateek Kumar Verma
2013CS10246

March 2015

Contents

1 Objectives **3**

2 Overall Design **3**

3 Sub Components **4**

3.1 Back End . . . 4

3.1.1 Alien . . . 4

3.1.2 Ship . . . 4

3.1.3 Color . . . 5

3.1.4 Bullet . . . 5

3.1.5 Board . . . 5

3.2 Artificial Intelligence . . . 6

3.3 Graphics . . . 6

3.4 Network Part . . . 6

3.4.1 Why UDP? . . . 6

3.4.2 Basic Network Design . . . 6

3.4.3 Network Outages . . . 7

3.4.4 Peer-to-Peer . . . 7

3.4.5 Basic UDP Functions . . . 7

4 Interaction amongst Sub Components **8**

4.1 Back-end and UI . . . 8

4.2 Back-end and Network . . . 8

5 Testing Of Components **8**

5.1 General Unit Tests . . . 8

5.2 Graphics . . . 8

5.3 Artificial Intelligence . . . 9

5.4 Network Component . . . 9

5.5 Overall Testing . . . 9

6 Extra Features **9**

6.1 Competitive Multi-player Mode . . . 9

6.2 3D Game-play . . . 9

6.3 Sound Effects . . . 9

6.4 Replacement of Player by AI . . . 9

1 Objectives

Design a game which is :

- Multi-player on-line without a central server.
- Has a artificial intelligence component.
- Is an action game and not a simple board game.

2 Overall Design

The game which we will build is space invaders. It involves the player controlling a space ship and shooting down aliens. The aliens will fight back with bullets and missiles. The player has a limited number of lives and has to score the maximum in them.

1. The application would be programmed in C++.
2. The GUI part would involve OpenGL.
3. UDP sockets will be used for network data transfer.
4. POSIX threads will be used to run the network and back-end in parallel.
5. Inter thread synchronization would be done using mutex lock.

3 Sub Components

3.1 Back End

TODO: FARAN

3.1.1 Alien

Listing 1: Class Parameters for Alien

```
class Alien
{
private:
    float XPos;             // X coordinate
    float YPos;             // Y coordinate
    float Angle;            // Orientation angle
    Color ColorOfAlien;     // Color
    int Level;              // AI difficulty level
    int PresentLives;       // Lives left
    int NumberBullets;      // Bullets fired per shot
    int NumberMissiles;     // Number of missiles left
    int AlienType;          // Type
};
```

3.1.2 Ship

Listing 2: Class Parameters for Ship

```
class Ship
{
private:
    float XPos;             // X coordinate
    float YPos;             // Y coordinate
    float Angle;            // Angle
    std::string Name;       // Name of player
    Color ColorOfShip;      // Color of ship
    int Lives;              // Lives left
    int Score;              // Score of player
    int Multiplier;         // Multiplying factor
    int Kills;              // No. of kills
    int Id;                 // Player id
    int NumberBullets;      // Bullets fired per shot
    int NumberMissiles;     // Number of missiles left
    int AILevel;            // Level of AI
};
```

3.1.3 Color

Listing 3: Class Parameters for Color

```
class Color
{
private:
    float R;                    // Value of R component
    float G;                    // Value of G component
    float B;                    // Value of B component
};
```

3.1.4 Bullet

Listing 4: Class Parameters for Bullet

```
class Bullet
{
private:
    float XPos;                 // X Coordinate
    float YPos;                 // Y Coordinate
    float VelX;                 // Velocity X
    float VelY;                 // Velocity Y
    Color ColorOfBullet;        // Color
    int ShipId;                 // Id of ship fired from
    bool TypeAI;                // If AI bullet
    bool TypePlayer;            // Player type
};
```

3.1.5 Board

Listing 5: Class Parameters for Board

```
class Board
{
private:
    std::vector<Ship> VectorShips;        // All ships
    std::vector<Bullet> VectorBullets;    // All bullets
    std::vector<Alien> VectorAliens;      // All aliens
    double DimensionPosX;                 // Dimensions + x
    double DimensionPosY;                 // Dimensions + y
    double DimensionNegX;                 // Dimensions - x
    double DimensionNegY;                 // Dimensions - y
};
```

3.2 Artificial Intelligence

TODO: KABIR

3.3 Graphics

TODO: KG

3.4 Network Part

We will use the User Datagram Protocol (UDP) to design and implement the network aspect of the assignment. It is a communications protocol that offers a limited amount of service when messages are exchanged between computers in a network that uses the Internet Protocol (IP). UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram. UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level.

3.4.1 Why UDP?

Our application is a real-time multi-player video game, thus we need fast transfer of data. As stated above, UDP can be fast but unreliable. Even if some of the packets are lost due to unreliability of UDP, it won't affect much. Things change so fast (i.e. player movement, bullets firing) in the game that it doesn't make sense to resend a lost packet as it will contain old information.

3.4.2 Basic Network Design

Our network will work on the basis of following points

- Each player will have two basic threads.
- One thread for receiving data from other players.
- Second thread for sending data to other players.

We shall be sending data (i.e. player position, AI components data etc) **as soon a frame is rendered**. This means that we will send almost 30-60 messages every second and hence the use of UDP is justified.

3.4.3 Network Outages

In the event of a network outages, we shall replace control of the ship of the lost player with AI of same level. The level of AI would be calculated on the basis of the score just before the player was lost. Once the player reconnects, he will automatically gain control of his lost ship.

3.4.4 Peer-to-Peer

Each of the player in the game would be exchanging its data with each and every other player. Thus we shall implement a peer-to-peer (P2P) connection. To control AI components of the game, we shall be using the concept of a "temporary server" or a "virtual server". A randomly chosen person will be chosen as the "temporary server". This player will act as the AI of the game and will send messages to all the other players accordingly. If a player other than this one disconnects, no change is required. If this player disconnects, another active player will be chosen to act as the "temporary server".

3.4.5 Basic UDP Functions

- Socket Creation

Listing 6: socket()

```
socket(AF_INET, SOCK_DGRAM, 0)
//AF_INET for Pv4 Internet protocol
//UDP uses SOCK_DGRAM
//0 is the protocol
```

- When a socket is created with socket(), it exists in a name space (address family) but has no address assigned to it. bind() assigns the address specified by addr to the socket referred to by the file descriptor sockfd. addrlen specifies the size, in bytes, of the address structure pointed to by addr. Traditionally, this operation is called "assigning a name to a socket".

Listing 7: bind()

```
bind(fd, (struct sockaddr *)&myaddr, sizeof(myaddr))
//fd is the sockfd
//myaddr is the address of the player
```

- Receiving data

Listing 8: recvfrom()

```
recvfrom(fd, buf, BUFSIZE, 0, (struct sockaddr *)&remaddr, &addrlen)
//fd is the sockfd
//buf is array of characters in which the message will be received
//remaddr is the address of the connection player
```

- Sending data

Listing 9: sendto()

```
sendto(fd, buf, strlen(buf), 0, (struct sockaddr *)&remaddr, addrlen)
//fd is the sockfd
//buf is array of characters which contains the data to be sent
//remaddr is the address of the sending player
```

4 Interaction amongst Sub Components

4.1 Back-end and UI

The GUI will use the data structures directly. The “Board” class will be available to generate the display on the screen. The user inputs from keyboard and mouse will be used to generate changes in the class object.

4.2 Back-end and Network

TODO KG

5 Testing Of Components

5.1 General Unit Tests

Listing 10: Class Parameters for Test

```
class Test
{
private:
    bool verbose;                //If test is to be conducted
    std::string description;     //String description of the test
    bool isPass;                 //Boolean if the test has passed
    void PrintPassFail(bool);    //Prints the status of the test
};
```

We will use the aforementioned class “Test” to perform unit tests on the different files created. This will ensure that all the functions work correctly against some test cases.

5.2 Graphics

To test the graphics component, we will create aliens and ships at chosen positions. The positions should change appropriately based on user inputs. Bullets should also be fired from the correct positions. When a bullet hits an alien or a ship, proper GUI effects will be added.

5.3 Artificial Intelligence

TODO KABIR

5.4 Network Component

Testing of network will be divided into two stages.

- In the first stage, we shall test the basic transfer of data. We shall be about 100 dummy messages from one player to another using a simple for loop. The receiver will record the number of messages received and thus we can determine the percentage of packets dropped.
- In the second stage, we shall be test the shifting of the “temporary server” from one player to another when the player gets disconnected. In this case, the “temporary server” shall be sending dummy messages. We shall connect about three players. After the connection is established, we shall close the application of the “temporary server”. One of the other two must become the “temporary server” and sending of dummy message should be continued.

5.5 Overall Testing

For overall testing of the game, we will play the game with as many players as possible and verify the smooth functioning of AI and network component. The network for some clients will be shut down suddenly to check if the transitions for the AI and others are smooth.

6 Extra Features

6.1 Competitive Multi-player Mode

In this multi-player mode, the players will be fighting each other. They will try to shoot each other down. Aliens will not be present in this mode. AI players will be present in this.

6.2 3D Game-play

The game will be taken to 3D in which the aliens and ships can be viewed in a 3D perspective. The camera position will be adjusted by the user based on input from mouse.

6.3 Sound Effects

Sound effects will be added for the entire game-play. When bullets are fired or some shoot down takes place, appropriate sounds will be played.

6.4 Replacement of Player by AI

In case of network failures, the player whose network has gone down will get replaced by an AI player of the same level as the player was. This will ensure completely seamless transition in case of network outages. Once the network comes back on, the player will replace the AI control.