BOOM!
Documentation

Wireless Game Developed For
INFO401

Nick Elder, Bing Leng, Duncan Meyer and Hui Zhang

Contents Page

System Manual

This section of the document discusses the code behind the Boom game. It covers the main functionality of the game from the setting up to the final end screen.

Game Setup

Three classes are required to setup a Boom Game. These are: `Boom`, `GameFrame` and `ActionArea`. These classes all interact with each other, along with the Framework's `ControlCenter` class to setup the basic structure of the game. This process is illustrated in Figure 1.

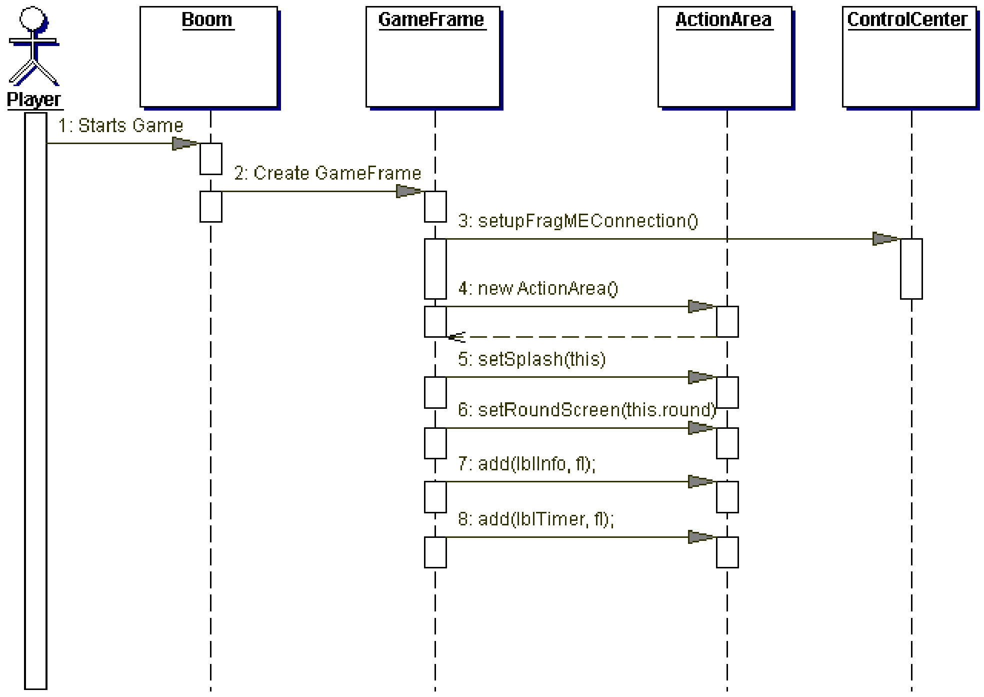


Figure 1: Setup Sequence Diagram

A player starts a game from the main class (`Boom`). The only functionality this class has is to create a new Game Frame.

```
game = new GameFrame();
```

The `GameFrame is where the main application code is done.` It creates the frame for the game as well as the status bar. A connection to the FRAGme framework is also setup. The framework takes care of all the connection stuff along with the FRAGme object and peer management. The connection setup is shown below.

```
ControlCenter.setUpConnections("Boom" +
GameSettings.verNum);
```

Once the connection is setup the `GameFrame` creates an `ActionArea` object. The `ActionArea` class is where the game play happens. The `ActionArea` sets up the gameSettings and landscape FRAGme objects. If no GameSettings or `Landscape` objects have been created, the `ActionArea` will create them as shown in Figure 2. Otherwise if a player is already in the game, the ActionArea uses the existing objects from the ControlCenter as shown in Figure 3. The following code is from the `ActionArea setupLandscape()` method.

```
     objs = ControlCenter.getAllObjects(Landscape.class);
     if (objs.size() == 1) {
       FMeObject obj = (FMeObject) objs.get(0);
       if (obj instanceof Landscape) {
         landscape = (Landscape) obj;
       }
      objs = ControlCenter.getAllObjects(GameSettings.class);
      obj = (FMeObject) objs.get(0);
      if (obj instanceof GameSettings) {
         gameSettings = (GameSettings) obj;
         gameSettings.setActionArea(this);
      }
     }
     else {
       firstPlayer = true;
       try {
         landscape = (Landscape)
ControlCenter.createNewObject(Landscape.class);
         landscape.createLandscape();
         gameSettings = (GameSettings)
ControlCenter.createNewObject(
             GameSettings.class);
         gameSettings.setActionArea(this);
       }
       catch (Exception e) {
         System.out.println("Problem creating lanscape
object");
         System.out.println(e.getMessage());
       }
     }
   }
```

Once the `ActionArea` has been created, the `GameFrame` then sets up some of the variables in the `ActionArea` required later in the game. These include the splash and round screens as well as the info and count down labels. After the initial setup is done the `Gameframe` waits with the splash screen showing for the player to select a tank.

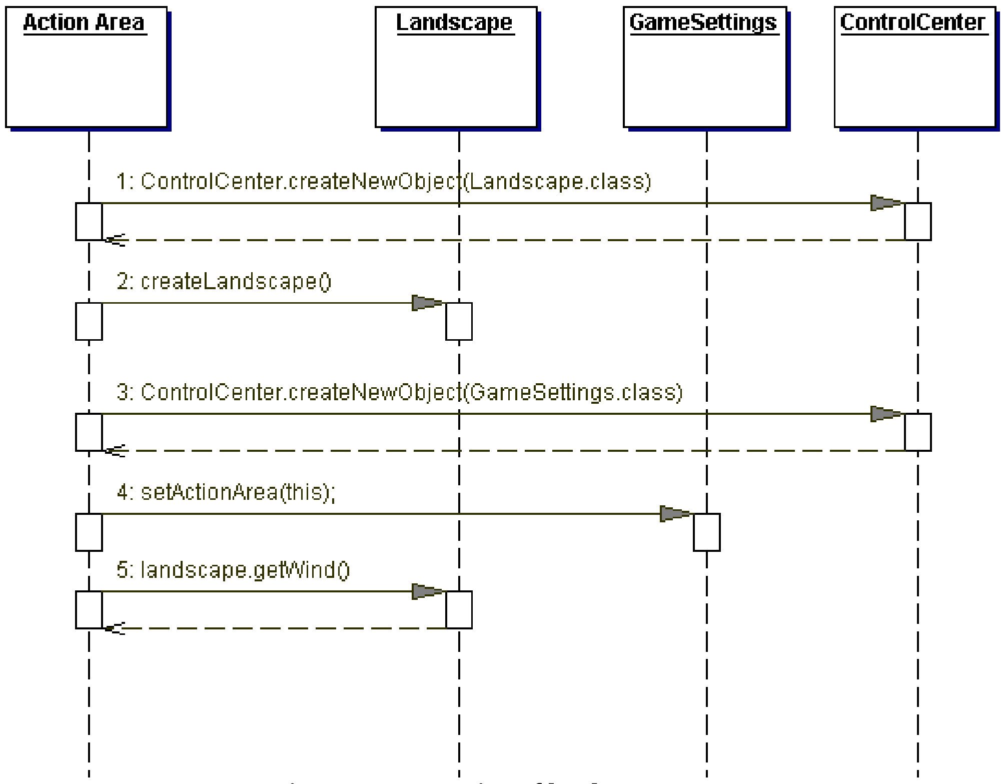


Figure 2: Generation of landscape

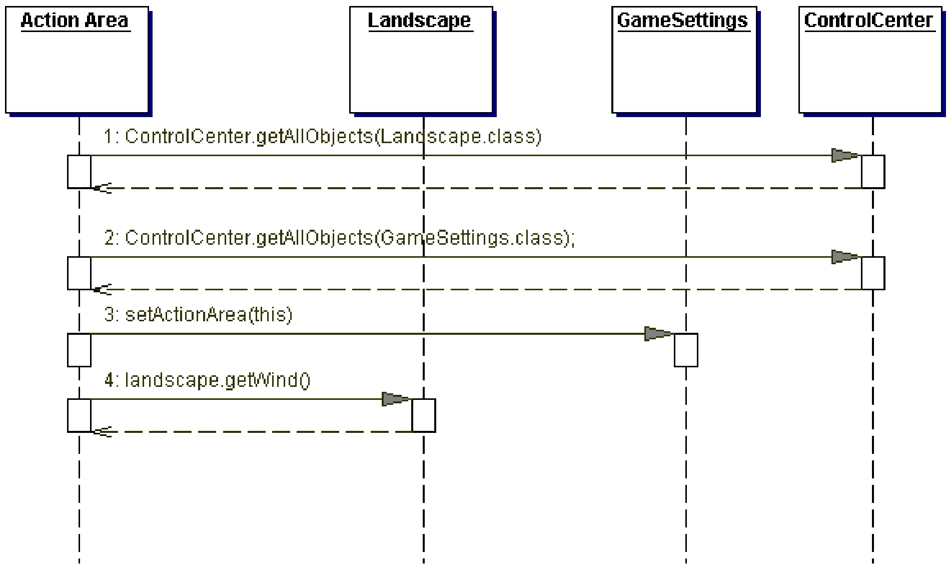

Action Area
Landscape
GameSettings
ControlCenter
1: ControlCenter.getAllObjects(Landscape.class)
2: ControlCenter.getAllObjects(GameSettings.class);
3: setActionArea(this)
4: landscape.getWind()

Figure 3: Getting already generated landscape

Creating your Tank

In Boom the tanks are all FRAGme objects and therefore the objects are all created on the framework side. Once a player selects a tank from the splash screen the `GameFrame` class creates the tank by calling the `createNewObject`() method on the ControlCenter.

```
    tank = (Tank) ControlCenter.createNewObject(Tank.class);
```

This method returns a FRAGme tank object. The `GameFrame` adds the tank to their `ActionArea` and sets up its initial values. After the values have been setup the tank is update for all the other peers by the tank.change() method.

```
        aa.setTank(tank);
        tank.setTankID(tankID);
        tank.setImage(i);
        tank.setImages(i);
        tank.setMyPeerName(ControlCenter.getMyPeerName());
        this.lblPlayerActual.
  setText(ControlCenter.getMyPeerName());
        aa.setMyTankID(tankID);
        aa.setLblTurnActual(this.lblTurnActual);
        aa.setLblScoreActual(this.lblScoreActual);
        aa.setLblRoundActual(this.lblRoundActual);
        aa.setLblWind(this.lblWindActual);
        aa.gameSettings.addPlayer();
        //Change FRAGME objects
        tank.change();
        aa.gameSettings.change();
        aa.getLandscape().change();
```

Once the player's tank has been created, the player then waits for any other players to join the game.

Starting actual game

During the initial setup the only FRAGme objects a player has a reference to are their own tank, the landscape and the game setting objects. Once a player chooses to start the game by, pressing the 'Enter' key, the players all iterator through the FRAGme objects looking for the other tanks. When they find one, they add it to their tank array in the action area. The tank array is then used whenever the tanks are used in the game for the rest of the round. This is done to keep network traffic as low as possible. The FRAGme method `getAllObjects`() is therefore only used during the setup process.

The following code is the code used to find all the tanks and add them to the `ActionArea` tank array. It is taken for the `getAllTanks()` method in the `GameFrame` class.

```
       objs = ControlCenter.getAllObjects();
       Iterator it = objs.iterator();
       while (it.hasNext()) {
       FMeObject obj = (FMeObject) it.next();

       if (obj instanceof Tank) {
         tempTank = (Tank) obj;
         aa.addTank( ( (Tank) obj));
       }
```

After all the tanks have been added to the tank array the game is ready to start. This is done by setting the gameStarted field in the game setting class to true and then tell all the other player by the following coded in the 'Enter' key event.

```
    aa.gameSettings.setGameStarted(true);
    aa.gameSettings.change();
```

Landscape generation

Before every round starts a new landscape is generated. The landscape is just an array of values for the height (y values) with the position in the array the x value.

The following code is the landscape algorithm.

```
int x = 0;
int y = 0;
double num;
double randomNum;

randomNum = Math.random();
y = (int) (randomNum * max);

while (x < width) {
randomNum = Math.random();
 // land is moving down
if (randomNum < .5) {
//how long it is going down
   randomNum = Math.random() * numInRow;
   while (randomNum > 0 && y > min && x < width) {
     num = Math.random();
     if (num > 0.5) {
        y = y - 1;
     }
     randomNum--;
     landscape[x] = y;
     landscapeLevel[x] = 1;
     x++;
  }
}
 // land is moving up
 else {
   randomNum = Math.random() * numInRow; //how long it is going up
   while (randomNum > 0 && y < max && x < width) {
      num = Math.random();
31       if (num > 0.5) {
32         y = y + 1;
33       }
34       randomNum--;
35       landscape[x] = y;
36       landscapeLevel[x] = 2;
37       x++;
38       }
39    }
40 }
41
42 super.change();
43 return landscape;
44 }
```

As shown by lines 6 and 7 of the landscape algorithm the algorithm starts by taking a random number (between 0 and 1) and multiplies it by the max height of the display. This returns the starting point of the landscape. The algorithm then goings into a while loop (lines 9 to 40) until it has generated the rest of the landscape. Each time through the while loop a new random number is taken. If the random number is less than 0.5 then the landscape moves down (lines 11 to 25). If the number is greater than 0.5 the landscape goes up (lines 27 to 40). After deciding whether to go up or down another random number is used to determine the number of x positions in arrow that the landscape is going up or down. It then uses another while loop for this number of times storing the y value in the array. Once this is complete the array of y values is used to paint the landscape which will be covered in a later section. The new landscape is then sent to the other peers by line 42.

Just before the landscape is created the wind for a round is generated. This is just done by using a random number and multiplying it by 10 as shown below.

```
wind = (int)(Math.random() * 10);
```

Tank Movement

A player moves a tank by pressing the left and right arrow. When one of these buttons is pressed, the GameFrame checks to see if the tank is already moving. If the tank is moving then the button pressed does nothing. When the tank is not moving it tells the tank which direction to move and then tells it to move. The tank setMove(true) method is call on the tank so the other peers know that the tank is moving and can sets their copy of the tank to move. It then calls the method moveTank(1) on the tank that moves the tank on their own peer. The parameter is the direction that the tank will move. If it is 1 the tank will move to the left and if 2 it will move to the right. This is shown in the following code from the GameFrame class.

```
if (tankMoving != true) {
        tankMoving = true;
        tank.setDirection(1);
        tank.setMove(true);
        tank.moveTank(1);
}
```

When the moveTank(int i) method is called on the tank a new TankMoving object is created. The TankMoving object is run in a new thread. The code from the moveTank method in tank is shown below.

```
public void moveTank(int i) {
    TankMoving tankmoving = new TankMoving(i, aa, this);
    tankmoving.start();
}
```

The run() method in the TankMoving thread moves the tank 10 times by calling the tank move method 10 times and paints the tank each time as shown in the code below.

```
  public void run() {
    for(int j = 0; j < 10; j++) {
      tank.move(i);
      aa.moveTank();
      try {
        sleep(50);
      }catch (Exception e) {
        System.out.println("tankmomving exception");
      }
    }
    GameFrame.tankMoving = false;
  }
```

The method below is the move method in the tank class.

```
  public void move(int i) {
    if (i == 1 && this.getX() <(aa.getLandscape().getWidth()
- 17)) {
      x = this.getX() + 3;
      y = Landscape.landscape[this.getX() + 1] - 17;
    }
    else if (this.getX() > 0) {
      x = this.getX() - 3;
      if(x <= 0) {
        x = 0;
      }
      y = Landscape.landscape[this.getX() + 1] - 17;
    }
  }
```

Depending on the direction value passed in as the parameter the new x and y values are set.

Painting

Painting on the Zaurus uses a large amount of memory and therefore can be very slow. To minimize this as much as possible Boom only paints the screen when something changes. When it does paint it only paints the part that has changed. This section will cover the different objects that are drawn and how it is done.

Landscape Paint

The landscape is only painted at the start of a game or round. After a round has started there is no need for the whole landscape to be repainted since it does not completely change. Part of the landscape does change when a tank moves or there is an explosion. In this case there is a modify method that is called to make that change. The landscape is first painted using the following code:

```
g.setColor(GameFrame.bgc);
g.fillRect(0,0, 640, 440);
g.setColor(new Color(0, 128,0));
for (int i = 0; i < 640; i++) {
  g.drawLine(i, landscape[i], i, 440);
}
```

The first two line of the code paint the whole action area with the background colour. This colour is stored in the GameFrame so it can be used in any part of the game. The for loop is where the landscape itself is painted. The loop works it way from the left to the right painting the landscape. For each x value a line is drawn from the y value in the landscape array to the bottom of the screen. For example, when x = 0 a line is drawn from coordinates (0, landscape[0]) to (0, 440). When all the lines are put together it gives an effect of the whole landscape being filled.

Tank Paint

The tanks are only painted at the start of a round and whenever a tank moves. The painting of the tanks only paints a clipping area around the tank and therefore, does not paint the whole screen. Since the landscape changes a little with the tank running over it an area around the landscape is repainted before the tank is painted. The code for this is shown below.

```
public void paint(int x, int y, Graphics g) {
   g.setClip(x-3,y-3,31,31);
   for(int xCoord = x-3; xCoord <= x + 31; xCoord++) {
     g.setColor(new Color(0, 128,0));
     if( xCoord > 0 && xCoord < this.width - 20){
       g.drawLine(xCoord, landscape[xCoord], xCoord, 440);
     }
   }
}
```

The paint method first sets the clipping area to the area around the tank. This means that only this area of the screen will be painted. The for loop then moves it way through this area of the screen repainting the landscape as it goes. Once this landscape has been updated the tank is then drawn in the new position. The code for painting the tanks is shown below.

```
 1   g.setClip(this.getX() - 5, this.getY() - 3, width + 3,
height + 3);
 2   g.setColor(bgc);
 3   g.fillOval(oldX - 2, oldY + 7, 20, 10);
 4   g.fillOval(oldX + 3, oldY + 2, 10, 6);
 5   g.setColor(tankColor);
 6   g.fillOval(getX() + 3, getY() + 2, 10, 6);
 7   g.setColor(Color.darkGray);
 8   g.fillOval(getX() - 2, getY() + 7, 20, 10);
 9   oldX = getX();
 10  oldY = getY();
```

Again the first line of code sets the clipping area to stop the whole screen being painted. Once this has been set the old tank is then painted over using the background colour in lines 2-4. Lines 5-7 then repaint the tank in the new position. After the tank has been paint the current x and y position are stored as oldX and oldY for the next time the tanks are painted.

Next Turn

Since the game is a turn based game it is important to keep the current players turn the same. For this reason the players turn is stored in the `GameSetting` class. Since this class is a FRAGEme object it is kept the same over all the other peers by calling the change() method. Each time a player tries to shoot (hits the fire key) a check is made on the `GameSetting` class to see whose turn it is. If it is the players turn they shot is made. If it is not then the key press is just thrown away.

A next call can be made in two different ways. One of ways is using the counter and the other is after a shoot has been made. When it is a player turn the counter counts down from 10 to 0 by the following code.

```
class TurnTask extends TimerTask {
   int numWarningBeeps = 100;
   ActionArea aa;

   public TurnTask(ActionArea aa) {
     this.aa = aa;
     aa.setTime(10);
   }

   public void run() {
     timeStarted = true;
     aa.time--;
     GameFrame.lblTimer.setText(String.valueOf(time));
     try {
       Thread.sleep(1000);
     }
     catch (Exception e) {
       System.out.println(e);
     }
     if (aa.time <= 0) {
       GameFrame.lblTimer.setText("");
       timeStarted = false;
       gameSettings.nextTurn();
       gameSettings.change();
       turn = false;
       this.cancel();
     }
   }
```

This is an inner class in the `ActionArea` class which goes through a while loop ten times sleeping each time for one second. The count down is only done on the local peer who currently has the turn. Once the counter gets to zero the `nextTurn()` method in the `GameSettings` class is called and then sent to the rest of the peer by the `GameSettings.change()`.

The second method is after the current player whose turn it is fires. After the fire key is pressed and the tank has fire the `nextTurn()` method is called on the `GameSetting` class and the other peers are told.

Once the `nextTurn()` method is called the following code is executed in the `GameSetting` class.

```

    // Find the next tanks ID
    while (count <= ActionArea.tankArray.size() &&
!nextTank) {
      count++;
      if (ActionArea.tankArray.size() == tankTurn) {
        nextTankID = 1;
      }
      else {
        nextTankID = tankTurn + 1;
      }

      tankTurn = nextTankID;
      it = ActionArea.tankArray.iterator();
      tankNumber = 0;

      while (it.hasNext()) {
        Tank t = (Tank) it.next();

        if (t.isVisible()) {
          tankNumber++;
          System.out.println("t visible");
            System.out.println("Next Tank ID: " +
nextTankID);
          if ( (t.getTankID() == nextTankID) &&
t.isVisible()) {
            System.out.println("inside t visible");
            System.out.println("inside Next Tank ID: " +
nextTankID);
            nextTank = true;
          }
        }
      }
    }
```

```
//Check to see if there is more than one tank still in the
game
    if (tankNumber < 2) {

      nextTank = false;
      it = ActionArea.tankArray.iterator();
      while (it.hasNext()) {
        Tank t = (Tank) it.next();
        if(t.isVisible()){
          t.addToScore(1000);
        }
        t.setVisible(false);
        t.change();
      }
    }
```

```
      if (nextTank) {
       ActionArea.lblTurn.setText(String.valueOf(tankTurn));
      } else {
       System.out.println("Round Over");
       round ++;
       aa.getLandscape().createLandscape();
       aa.getLandscape().change();
       roundOver = true;
```

```
if(round > 3){
         aa.add(new GameOver(aa.tankArray));
         aa.repaint();
       } else{
         aa.preRoundOver();
         aa.roundOver();
       }
      }
```

The first part of the method goes through all the tanks in order until it finds the next possible tank. If the tank is still visible then it is set as the next tank. If not it continues through the tanks until it finds one.

The next section of code checks to see if there is more than one tank still in the game. If there are one or less tanks then the nextTank variable is set to false.

If there is more then one tank the third section just changes the round label. If not then it displays the end game screen.

The final section tells the other that the next turn has been set and to display the round over screen. This will be cover in more detail in the next section.

Round Over

After a round is over the roundOver() method is called in the ActionArea class. This method cancels the timer and resets the landscape and tank positions. It also displays the round over screen as shown below.

```
  public void roundOver() {
    turnTimer.cancel();
    System.out.println("in round over");
    roundScreen.setTankArray(this.tankArray);
    setTT();
    roundScreen.setVisible(true);
    roundScreen.repaint();

    resetTankStartingPoint();
    myTank.setVisible(true);
    myTank.change();
    paintLandscape();
  }
```

When one of the peers presses 'Enter' on the round over screen a new round starts. This is done by calling the newRound() method on the ActionArea class. This method paints the new landscape and sets the new round variables. The code is shown below.

```
public void newRound() {

    gameSettings.setRoundOver(false);
    gameSettings.setNewRound(false);
    gameSettings.change();
    roundScreen.setVisible(false);
    paintLandscape();

  }
```

End of Game

When the round field in the GameSettings class gets set to the round limit after a turn the end game screen is shown.

The end game screen shows multiple explosions on the screen before showing the scores of the players.

To get multiple explosions running on the game over splash screen concurrently, we have created a new class (logo) which extends Thread inside the GameOver class. GameOver class has a paint method which displays players, and their scores, inside the Logo class there is another paint method (they are on the same graphic context) which displays multiple explosions concurrently. It is quite flexible to add more graphic in, e.g. bird flying.

Here below is some sample code:

```
  public void paint(Graphics g) {
    this.g = g;
  ……
    Logo logo = new Logo();
    logo.run();
    ..........
    g.setFont(normalFont);
    while (it.hasNext()) {
      Tank t = (Tank) it.next();
     .......
      g.drawString("Tank: " + t.getTankID(), 175 , 150 +
(t.getTankID() * 20));
      g.drawString("Player: " + t.getMyPeerName(),280 , 150
+ (t.getTankID() * 20));
      g.drawString("Score: " + t.getScore(), 415, 150 +
(t.getTankID() * 20));
    .........
    }
  }
  public void update(Graphics g) {
    paint(g);
  }

  class Logo
      extends Thread {
    public void paint(Graphics g) {
      ....................
        g.setColor(Color.yellow);
        g.fillOval(200, 200, width, height);
        g.setColor(Color.white);
        g.fillOval(x + 10, y + 10, width - 20, height - 20);
   ................
    }
    public void run() {
      paint(g);
    }
  }
}
```

Boom User Guide

Overview of the Game

Boom is a multiplayer turn style arcade game that is loosely based on 80’s classic Scorched Earth. The game is designed to be played by two or more players each running their own copy on their own mobile device. The style of play is meant to be simple and fun so that people can enjoy a quick game over a wireless connection.

Loading the Game:

- Start the Zaurus.
- Launch *Embedded Konsole* from *start menu.*
- From the command line type*: cvm –jar boomver0.60.jar* to launch Boom.

Objective:

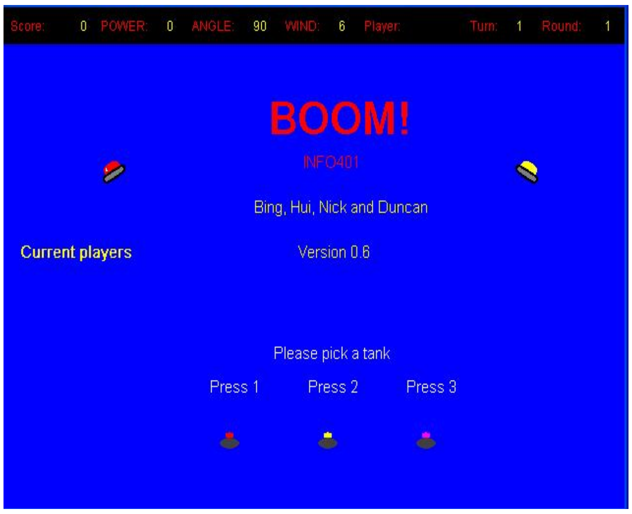
Boom is a multiplayer shooting game running in the peer-peer environment. With many unique types of vehicles, we open this game to you, so you can develop crafty and even sneaky new strategies. Each round has a unique map. The wind also changes the gravity and direction on each round, making it necessary to recalculate how the atmosphere will alter how you fire. It’s important to use your weapon well and adapt quickly, or you will surely lose quickly. Each turn, you will have a lot of different options available. You can shoot, or you can move. All of these things make the game dynamic, and help the game come alive for players. And if you win, you will receive certain points. Enjoy the Boom. Here below are rules of this game and how the score is calculated.

A	Move the barrel of the vehicle towards right
S	Move the barrel of the vehicle towards left
Arrow up	Increase Power
Arrow down	Decrease Power
Arrow right	Move the vehicle to the right
Arrow left	Move the vehicle to the left
Space bar	Launch the Missile
Hit a tank	Score +500
Blow your own tank	Score -500
Last one in the round	Score +1000

How to Play

These screenshots will help you to go through this game based on the two players scenario (PlayerA, PlayerB) and tell you how to play Boom.

Start a game


Score: 0 POWER: 0 ANGLE: 90 WIND: 6 Player: Turn: 1 Round: 1

BOOM!

INFO401

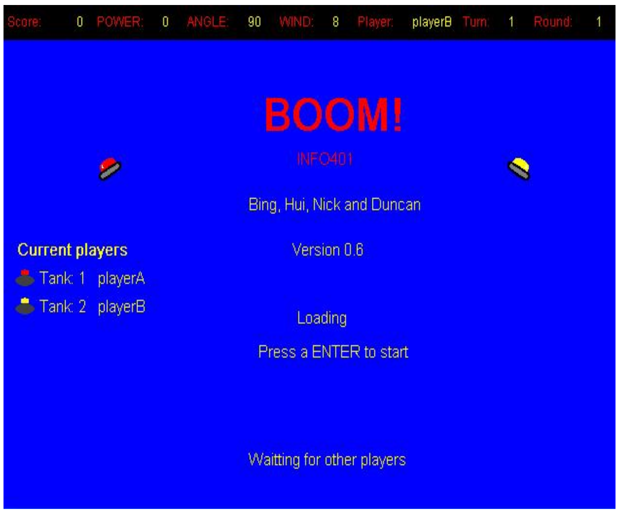
Bing, Hui, Nick and Duncan

Current players

Version 0.6

Please pick a tank

Press 1 Press 2 Press 3


Score: 0 POWER: 0 ANGLE: 90 WIND: 8 Player: playerB Turn: 1 Round: 1

BOOM!

INFO401

Bing, Hui, Nick and Duncan

Current players

Tank: 1 playerA

Tank: 2 playerB

Version 0.6

Loading

Press a ENTER to start

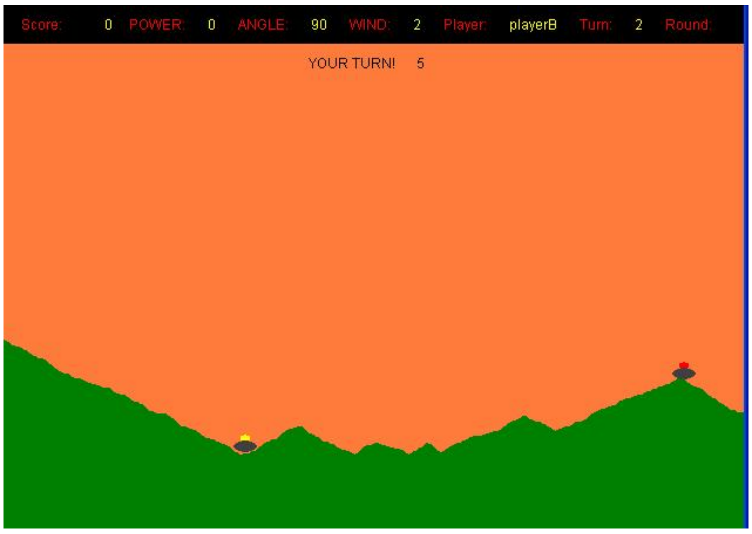
Waitting for other players

Figure 1 **Figure 2**

When the players start Boom, they will go to the entry screen (**Figure 1**), from here they will have a chance to pick a tank by pressing ***1*** or ***2*** or ***3***. After that they will go straight to the waiting screen (**Figure 2**), which tells how many players are willing to play this game. When all the players are ready to rock and roll, the first player presses the ***Enter*** key, which will lead them to the playing screen (**Figure 3, Figure 4**).

Note: One player could not run this game, it requires at least two players.

Moving


Score: 0 POWER: 0 ANGLE: 90 WIND: 2 Player: playerB Turn: 2 Round:

YOUR TURN! 5

Figure 3

In the Playing screen (**Figure 3**), the player can move their vehicle along the landscape. Press ***Arrow left*** to move to the left or ***Arrow right*** to move to the right. Wind is 2 currently and its direction is to the left, as you can see on the top of the screen.

Firing and Aiming

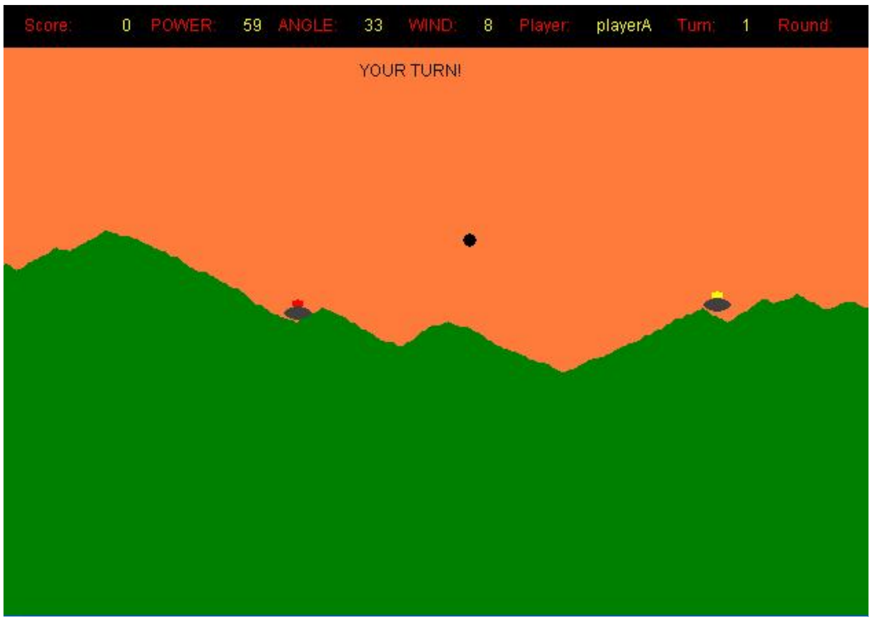

Score: 0 POWER: 59 ANGLE: 33 WIND: 8 Player: playerA Turn: 1 Round:
YOUR TURN!

Figure 4

In the playing screen (**Figure 4**), players can adjust their barrel angle and power, by pressing ***A*** to adjust angle towards the right, ***S*** to adjust angle towards the left, ***arrow up*** to increase power and ***arrow down*** to decrease power. After that you just press ***space bar*** to launch the missile.

Note: You only have ten seconds to adjust your angle and power.

Round Over

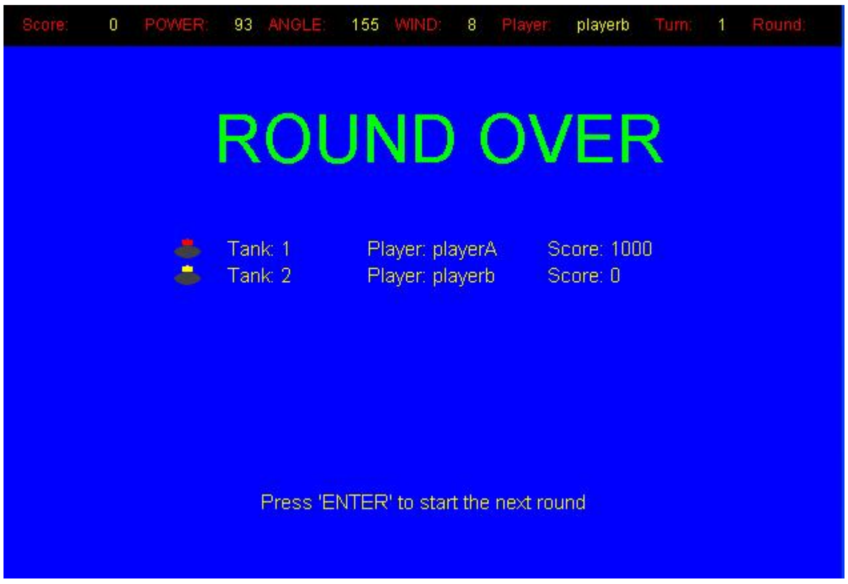

Score: 0 POWER: 93 ANGLE: 155 WIND: 8 Player: playerb Turn: 1 Round:
ROUND OVER
Tank: 1 Player: playerA Score: 1000
Tank: 2 Player: playerb Score: 0
Press 'ENTER' to start the next round

Figure 5

In the Round Over screen (**Figure 5**), you can see your current score. Also you can press ***Enter*** to start the next round. The next round will automatically generate another landscape. After three rounds you will be led to the winning screen (**Figure 6**).

Winning

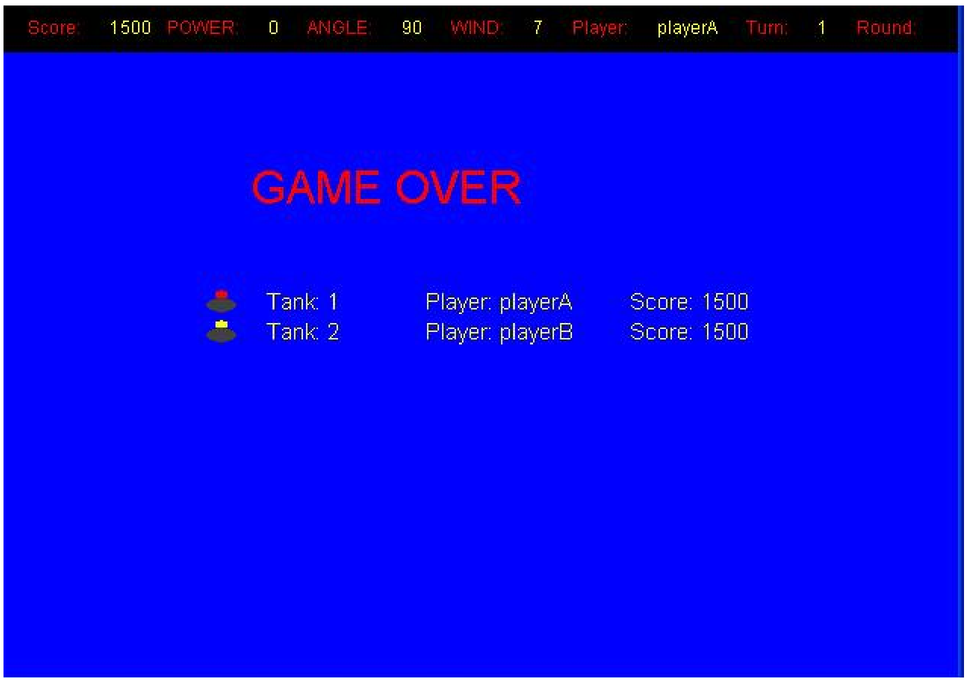

Score: 1500 POWER: 0 ANGLE: 90 WIND: 7 Player: playerA Turn: 1 Round:
GAME OVER
Tank: 1 Player: playerA Score: 1500
Tank: 2 Player: playerB Score: 1500

Figure 6

In the winning screen, you would see all the players score. Game Over.

Testing:

We decided that the best way to test Boom was to perform functional testing and make sure that the requirements of the game have been satisfied. We will define the requirement being tested and then detail the outcome.

Functionality Testing:

1. Starting a game:

Actions:
Player 1 starts the game and selects a tank.
Player 2 then starts the game and selects a different coloured tank.
Expected Outcome:
Both players appear in the start screen with their names next to their selected tank.
Actual Outcome:
Both player's names and tanks are present
Result:
PASS

2. Moving a Tank:

Actions:
Make a tank move left and right.
Expected Outcome:
The tank moves correctly across the map terrain.
Actual Outcome:
The tank moves along the terrain, but is sometimes appears slightly above the ground level.
Result:
PASS

3. Firing a missile:

Actions:
Player 1 fires a missile
Expected Outcome:
A missile appears and flies across a trajectory.
Actual Outcome:
Missile appeared and flew correctly.
Result:
PASS

4. Missile Exploding:

Actions:
Fire a missile and let it hit the ground.
Move tank over terrain to see if it has been altered.
Expected Outcome:
An explosion animation is shown and the terrain is altered.

Actual Outcome:
The explosion animation began when the missile hit the ground. The tank moves over the newly altered terrain. The tank clipping does not always work when the ground is very steep.
Result
PASS

5. Firing a missile out of screen:
Actions:
Fire a missile with power set high enough so it flies out of the visible screen and enters again.
Expected Outcome:
Missile is fired and leaves the screen, then enters again an explodes on impact with the ground.
Actual Outcome:
Missile is fired and leaves the screen, re-enters and correctly explodes on impact with the ground.
Result:
PASS

6. Landscape Generation:
Actions:
Begin a new round.
Expected Outcome:
A new landscape is generated correctly.
Actual Outcome:
The landscape was generated correctly.
Result:
PASS

7. New Round:
Actions:
Begin a new round and determine if tanks are set up correctly.
Expected Outcome:
A new landscape is generated and player tanks are positioned randomly on the map.
Actual Outcome:
The landscape is generated correctly but an extra tank is drawn on the landscape. This extra tank is an error in the painting because it does not exist other than on the screen.
Result:
FAIL

8. Destroying of Opponent Tank:
Actions:
Fire a missile and hit the opponent tank.
Expected Outcome:
The opponent tank explodes and he is removed from the round.
Actual Outcome:

The opponent tank explodes and disappears. He takes no further part in the round.
Result:
PASS

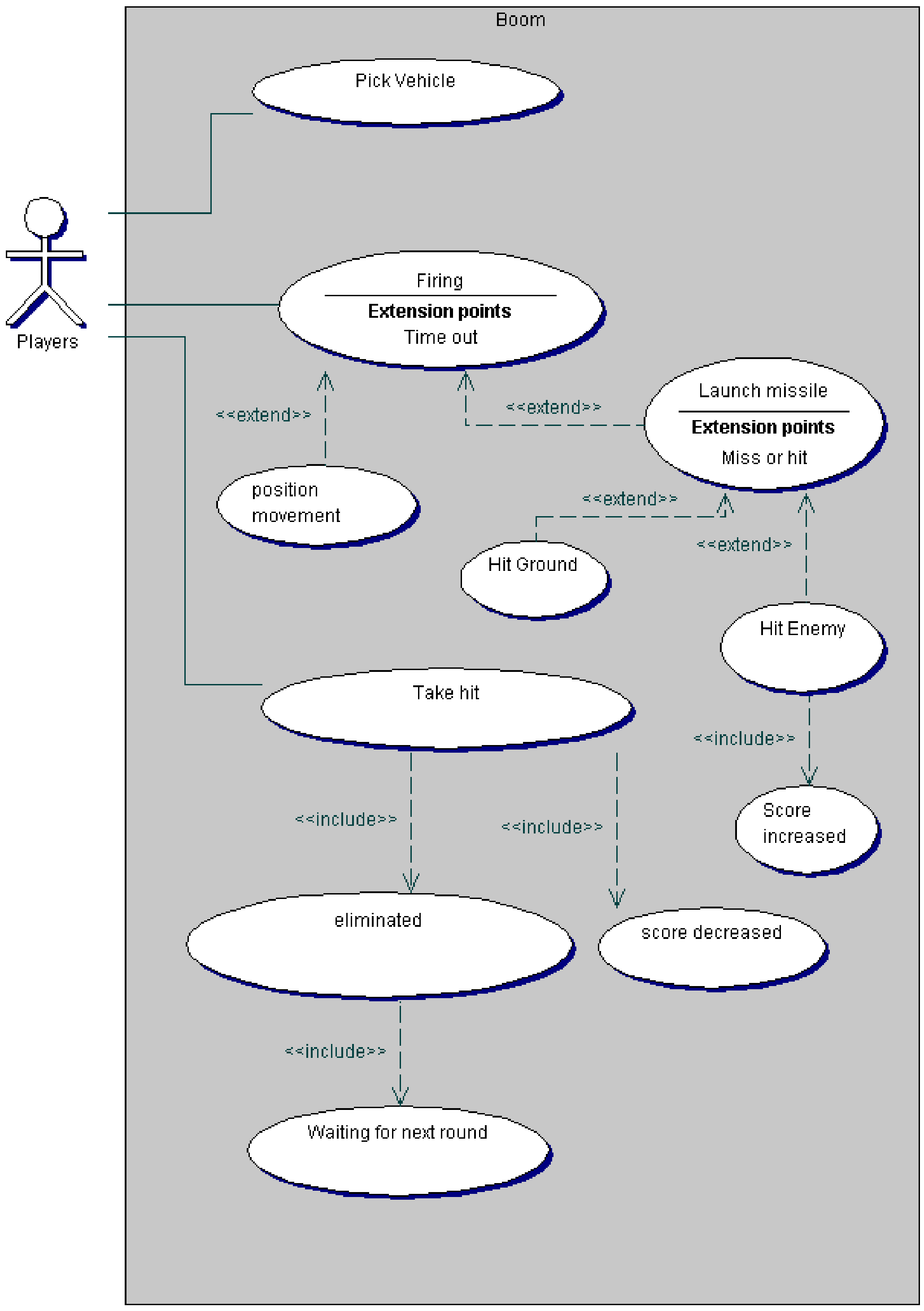
9. Destroy own Tank.
Actions:
Fire a missile with no power.
Expected Outcome:
The player's tank explodes and the opponent wins the round. 500 points are taken from the player's score.
Actual Outcome:
The players tank exploded and the correct number of points are taken from the player.
Result:
PASS.

10. Destroying Both Tanks:
Actions:
Move two tanks close to each other and fire a missile that destroys both.
Expected Outcome:
Both tanks should explode and the round should end with both players losing 500 points.
Actual Outcome:
The round did not end, the game locks in a deadlock situation.
Result:
FAIL.

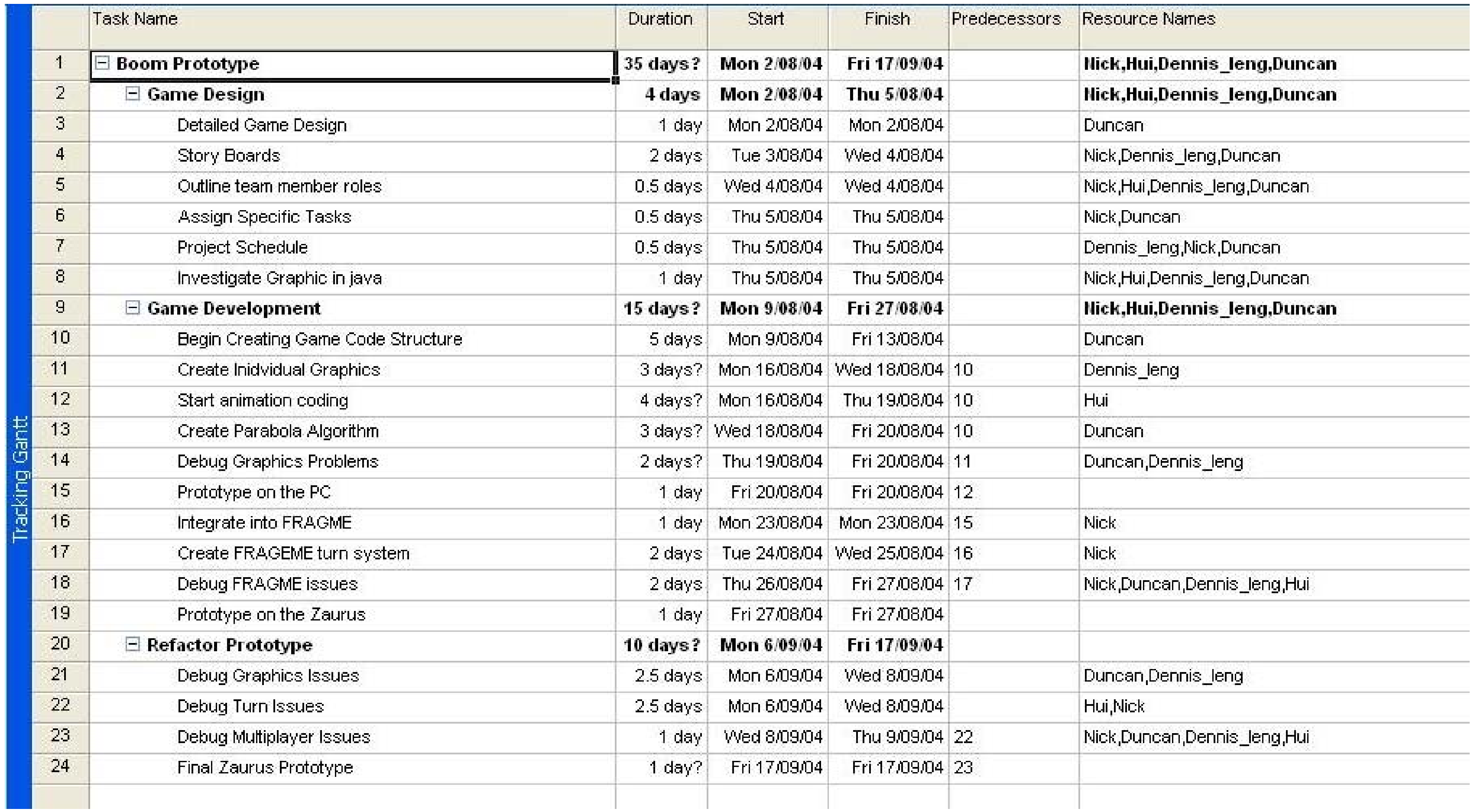
11. Firing when it is not your turn:
Actions:
Press the fire button when it is not your turn.
Expected Outcome:
The player should not be able to fire a missile.
Actual Outcome:
No missile is fired.
Result:
PASS.

12. Firing with no power:
Actions:
Decrease power to zero.
Expected Outcome:
No missile should be fired. The player should keep his turn.
Actual Outcome:
A missile is fired and destroys the player's tank.
Result:
FAIL.

Appendix A: Use Case diagram

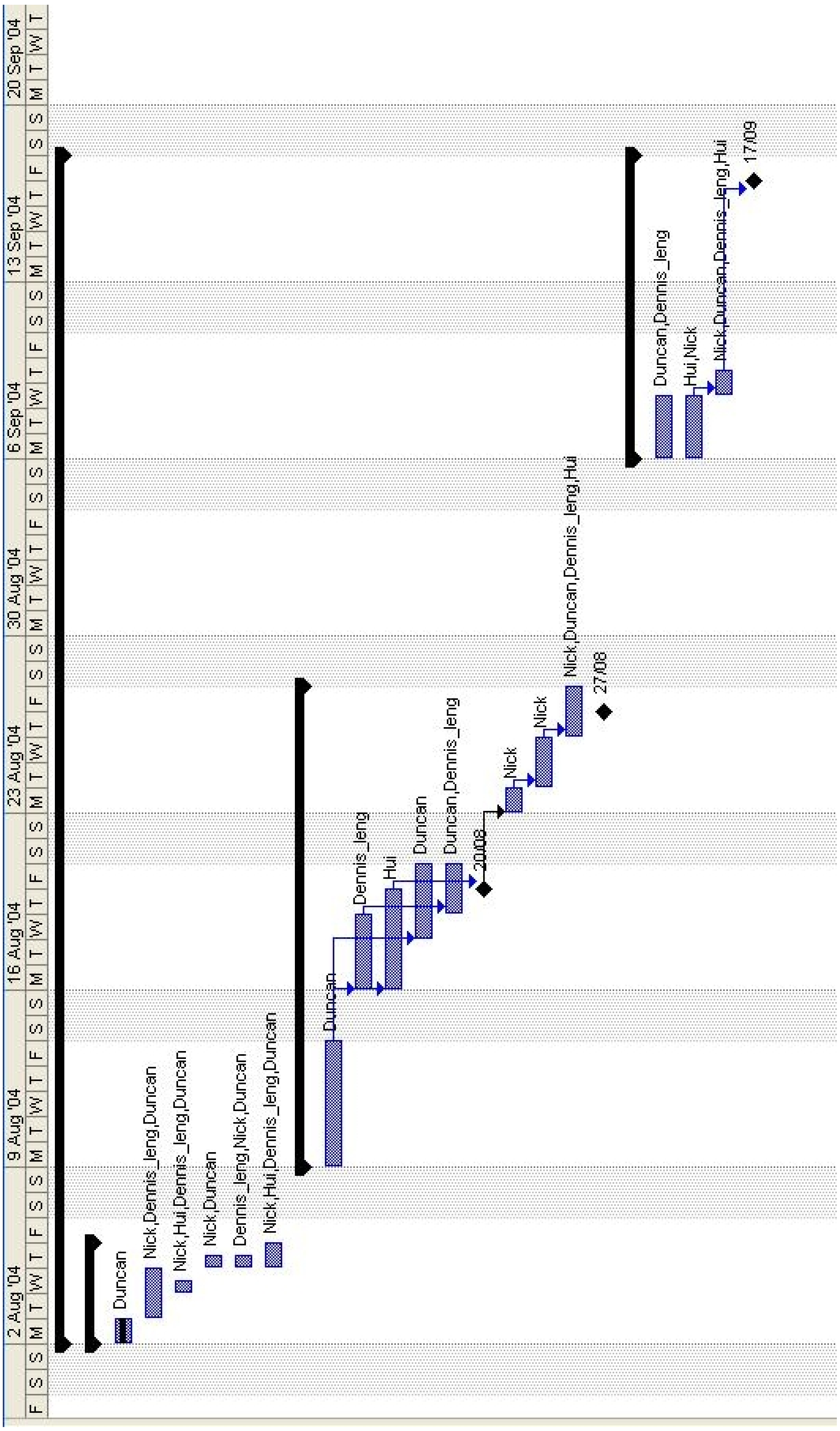


Appendix B: MS Project



Tracking Gantt

	Task Name	Duration	Start	Finish	Predecessors	Resource Names
1	⊟ **Boom Prototype**	**35 days?**	**Mon 2/08/04**	**Fri 17/09/04**		**Nick,Hui,Dennis_Ieng,Duncan**
2	⊟ **Game Design**	**4 days**	**Mon 2/08/04**	**Thu 5/08/04**		**Nick,Hui,Dennis_Ieng,Duncan**
3	Detailed Game Design	1 day	Mon 2/08/04	Mon 2/08/04		Duncan
4	Story Boards	2 days	Tue 3/08/04	Wed 4/08/04		Nick,Dennis_Ieng,Duncan
5	Outline team member roles	0.5 days	Wed 4/08/04	Wed 4/08/04		Nick,Hui,Dennis_Ieng,Duncan
6	Assign Specific Tasks	0.5 days	Thu 5/08/04	Thu 5/08/04		Nick,Duncan
7	Project Schedule	0.5 days	Thu 5/08/04	Thu 5/08/04		Dennis_Ieng,Nick,Duncan
8	Investigate Graphic in java	1 day	Thu 5/08/04	Thu 5/08/04		Nick,Hui,Dennis_Ieng,Duncan
9	⊟ **Game Development**	**15 days?**	**Mon 9/08/04**	**Fri 27/08/04**		**Nick,Hui,Dennis_Ieng,Duncan**
10	Begin Creating Game Code Structure	5 days	Mon 9/08/04	Fri 13/08/04		Duncan
11	Create Inidvidual Graphics	3 days?	Mon 16/08/04	Wed 18/08/04	10	Dennis_Ieng
12	Start animation coding	4 days?	Mon 16/08/04	Thu 19/08/04	10	Hui
13	Create Parabola Algorithm	3 days?	Wed 18/08/04	Fri 20/08/04	10	Duncan
14	Debug Graphics Problems	2 days?	Thu 19/08/04	Fri 20/08/04	11	Duncan,Dennis_Ieng
15	Prototype on the PC	1 day	Fri 20/08/04	Fri 20/08/04	12	
16	Integrate into FRAGME	1 day	Mon 23/08/04	Mon 23/08/04	15	Nick
17	Create FRAGEME turn system	2 days	Tue 24/08/04	Wed 25/08/04	16	Nick
18	Debug FRAGME issues	2 days	Thu 26/08/04	Fri 27/08/04	17	Nick,Duncan,Dennis_Ieng,Hui
19	Prototype on the Zaurus	1 day	Fri 27/08/04	Fri 27/08/04		
20	⊟ **Refactor Prototype**	**10 days?**	**Mon 6/09/04**	**Fri 17/09/04**		
21	Debug Graphics Issues	2.5 days	Mon 6/09/04	Wed 8/09/04		Duncan,Dennis_Ieng
22	Debug Turn Issues	2.5 days	Mon 6/09/04	Wed 8/09/04		Hui,Nick
23	Debug Multiplayer Issues	1 day	Wed 8/09/04	Thu 9/09/04	22	Nick,Duncan,Dennis_Ieng,Hui
24	Final Zaurus Prototype	1 day?	Fri 17/09/04	Fri 17/09/04	23	


2 Aug '04
9 Aug '04
16 Aug '04
23 Aug '04
30 Aug '04
6 Sep '04
13 Sep '04
20 Sep '04
Duncan
Nick,Dennis_Ieng,Duncan
Nick,Hui,Dennis_Ieng,Duncan
Nick,Duncan
Dennis_Ieng,Nick,Duncan
Nick,Hui,Dennis_Ieng,Duncan
Duncan
Dennis_Ieng
Hui
Duncan
Duncan,Dennis_Ieng
20/08
Nick
Nick
Nick,Duncan,Dennis_Ieng,Hui
27/08
Duncan,Dennis_Ieng
Hui,Nick
Nick,Duncan,Dennis_Ieng,Hui
17/09