CS4303 Video Games Practical 1

120008200

October 1, 2015

1 Introduction

The practical has been completed, meeting the basic requirements including:

- Particles (rendered as circles) fall from the sky with a random initial velocity, affected by both gravity and drag.
- Quantity and initial velocity of particles vary by round.
- A number of cities occupy the ground which the player must defend.
- If a particle hits a city, it is destroyed. If all cities are destroyed, the game is over.
- The player controls a crosshair to fire missiles. When a missile is fired, the blast radius dynamically changes to simulate an explosion. Any particles colliding with the blast radius are destroyed.
- The game is organised into a number of rounds, each increasing in difficulty.

A number of extensions have been completed including:

- Particles will sometimes split into several child particles.
- Bombers fly across the screen and drop additional particles.
- The flight of fired missiles is visualised. Missiles are fired from two batteries, located at each side of the screen.
- Sound effects have been included in the game.

2 Design

2.1 Game phases

The game has been split into four distinct phases (or states):

- NotStarted - the entry point for the game
- InRound - the player is defending cities

- BetweenRounds - a short interval between rounds
- Over - all cities have been destroyed

The main game loop switches over the current game phase, and updates/visualises objects as required.

```
case GameState.NotStarted:
  visualise.startScreen();
  visualise.crosshair();
  break;

case GameState.InRound:
  round.update();
  visualise.world();
  visualise.cities(cities);
  visualise.particles(round.particles);
  visualise.bombers(round.bombers);
  visualise.missiles(round.missiles);
  visualise.statistics(round.number, points, round.missilesRemaining);
  visualise.crosshair();
  break;

case GameState.BetweenRounds:
  visualise.betweenRounds(round.number + 1, points);
  betweenRoundsTimerTick();
  break;

case GameState.Over:
  visualise.statistics(round.number, points, round.missilesRemaining);
  visualise.gameOver();
  visualise.crosshair();
  break;
```

Figure 1: The main game loop, split by game phase

3 Collision detection

3.1 Circles

Particles and missiles are rendered as circles, making collision detection between the two fairly simple. If the distance between the two vectors is less or equal to the sum of their radii, a collision is detected.

```
distance = PVector.dist(p.position, m.position);
sumOfRadiuses = p.diameter/2 + m.diameter/2;

if(distance <= sumOfRadiuses)
{
  points += 10;
  collisions.add(p);
  sound.missileHit();
}
```

Figure 2: Collision detection between particle p and missile m

3.2 Blocks

Blocks (or rectangles) require slightly more effort to detect collisions. Cities and Bombers are rendered using images, and for the purposes of collision detection they are assumed to be rectangles.

To avoid repetition, *City* and *Bomber* extend an abstract *Block* class which aids collision detection.

```
public boolean isHit(PVector pos, float radius)
{
  if( pos.x + radius > xMin
       && pos.x - radius < xMax
       && pos.y + radius > yMin
       && pos.y - radius < yMax)
       {
         return true;
       }

  return false;
}
```

Figure 3: Check for a missile hit in the block class

4 Soring

A player scores points by:

- Destroying missiles (10)
- Destroying bombers (50)

End of round bonuses are awarded for surviving cities and unused missiles. A destroyed city is also rebuilt on completion of a round.

5 Screenshots

Figure 4: Start screen

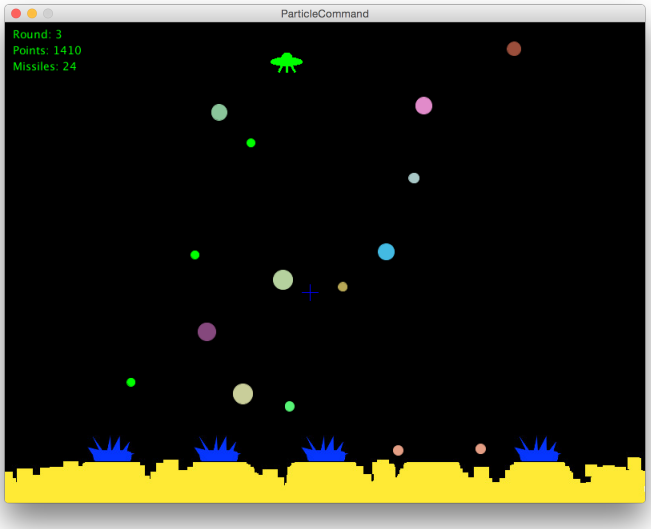

Figure 5: Mid round

Figure 6: Between rounds

Figure 7: Game over

6 References

In-game sound effects are creative-commons licensed sounds, **not** my own work.

Game graphics (background, bombers, cities) are my own work.