Overview:

The aim of this practical was to gain practical experience of using Processing language and implement some of the concepts in the Physics component of the module. Such aims were to be achieved through developing a “Particle Command” game - a variant of the video game “Missile Command” with particular requirements. Gameplay elements, as well as design decisions will be discussed in ***design*** section.

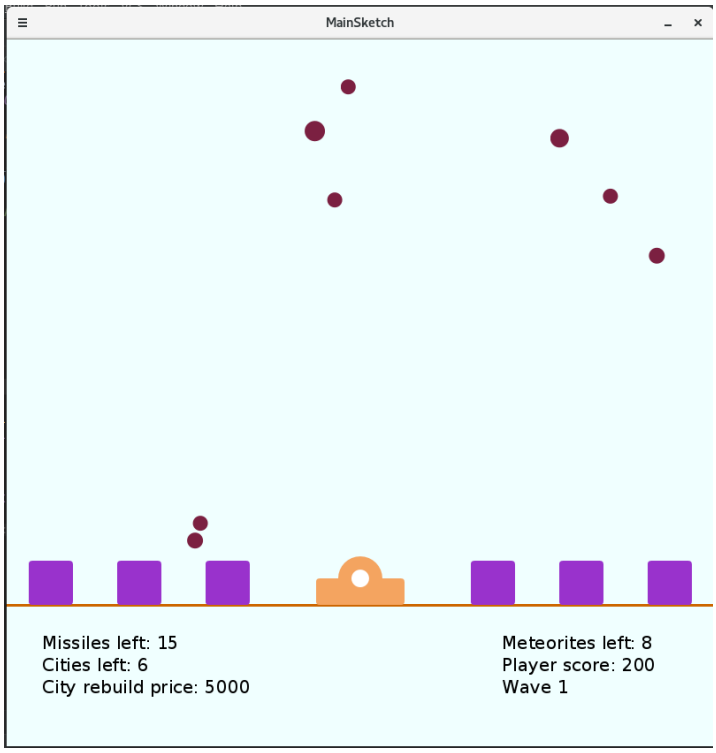
In order to run the programme, open the project in terminal and type “./run.sh”.

Design:

This section begins with the description of “general game loop” and then would go into details of each particular feature and the reasoning behind it.

General game loop:

The game is organised into “waves”. Player starts at wave 1, with default amount of particles that appear “in the sky” and fixed amount of missiles and cities on the ground. Each wave lasts until the last particle is destroyed or until the last city is destroyed. Wave summary is displayed at the end of each wave with the ability for a player to recover any destroyed cities, provided he/she has sufficient amount of points (points are used as currency, with the recovery amount deducted from total player score). Player has to hit “Enter” button to proceed to the next level, which has increased amount of particles and missiles, as well as slightly stronger gravity and particle appearance rate. Whenever all cities are destroyed – a summary message is displayed, asking the player to either start a new game or quit the programme by pressing specified buttons.



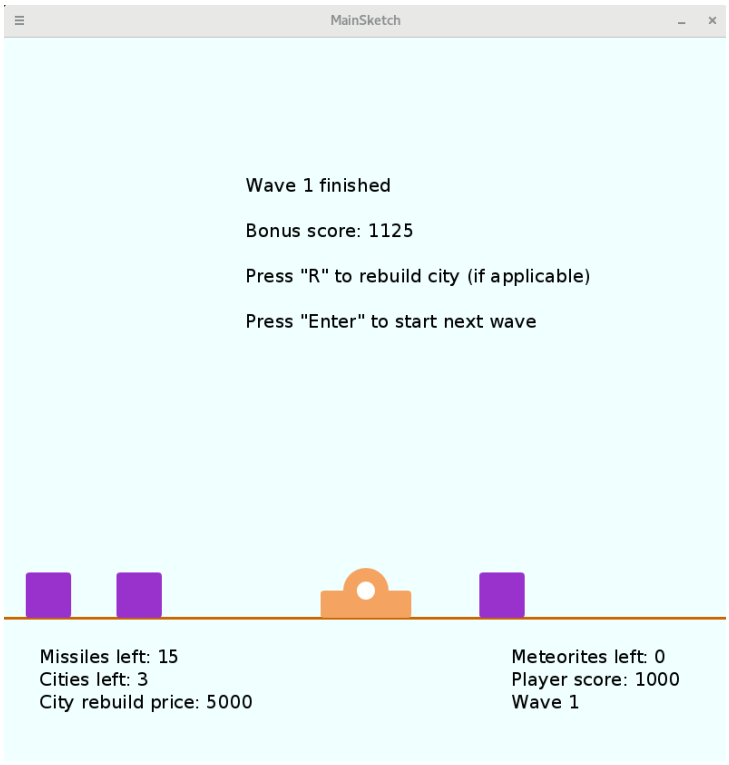


Fig.1 First wave

Fig.2 Game state between waves

Particle:

In order to represent a particle I have created a Particle class. The structure of the class was inspired by the examples from the lecture, yet modified to suit the needs of my project. For example, I have added logic to represent particle destruction and detect a collision with the city.

```
void integrate(PVector gravity) {
    if (invMass <= 0f) return;

    position.add(velocity);
    PVector acceleration = gravity.copy();
    acceleration.mult(invMass);

    velocity.add(acceleration);
    velocity.mult(DAMPING);
    if (position.x < 0 || position.x > parent.width)
        cease_existence = !cease_existence;
    if (position.y < 0 - 50 || position.y > ground - size * 0.5)
        cease_existence = !cease_existence;
    detectCityCollision();
}

void displayParticle() {
    parent.fill(parent.color( v1: 123, v2: 32, v3: 65));
    parent.ellipse(position.x, position.y, size, size);
    parent.noFill();

}

void detectCityCollision() {
    for (int i = 0; i < cities.size(); i++) {
        City city = cities.get(i);
        if (!city.isDestroyed()) {
            if ((position.x >= city.xpos - size/2 && position.x <= city.xpos + city.SIZE + size/2)) {
                if (position.y + size/2 >= city.ypos && position.y+ size/2 <= city.ypos + city.SIZE) {
                    cease_existence = !cease_existence;
                    city.setDestroyed(true);
                }
            }
        }
    }
}
```

Fig.3 Particle implementation with modifications

My particles are affected by gravity and drag (I didn't quite understand the need of extensive drag force generator, so I have used Millington to simulate drag). The

amount of particles was decided to be 10 for the first wave and then increased by 5 per each consecutive wave. Such numbers were chosen so the game feels easy yet challenging enough to play.

```
private void newGame() {
    draw_counter = 0;
    player_score = 0;
    wave_particles_amount = STARTING_AMOUNT_PARTICLES;
    wave_particles_counter = wave_particles_amount;
    city_rebuild_price = 5000;
    particle_frequency = 90;
    wave_counter = 1;
    n_missiles = new Double( value: wave_particles_counter * 1.5f).intValue();
    particles = new ArrayList<>();
    existing_cities = new ArrayList<>();
    destroyed_cities = new ArrayList<>();
    missiles = new ArrayList<>();
    gravity = new PVector( x: 0f,  y: 1.3f);
    contactResolver = new ContactResolver();
    missile_contacts = new ArrayList<>();
    int building_xstep = 25;
    for (int i = 0; i < CITIES_AMOUNT; i++) {
        existing_cities.add(new City( p: this,
                building_xstep,
                y: SURFACE_YPOS - CITY_SIZE,
                CITY_SIZE,
                color( v1: 153, v2: 50, v3: 204)
        ));
        if (i == 2) building_xstep += 200;
        building_xstep += 100;
    }
    font = createFont( name: "Arial",  size: 20,  smooth: true);
    cursor(CROSS);
    createGun();

    end_game = false;
    playing = true;
}
```

Fig.4 Setting initial values

Ground&cities:

I have decided to go with the Missile Command -like layout, with one exception: I have only one gun in the middle and it cannot be destroyed by the falling particles. I have a total of 6 cities, which are equally distributed on the land in bulks of three with a gun in the middle. I think that such an amount of cities is sufficient enough to give the player some flexibility in terms of missed particles and the equal distribution of the cities on the ground preserves equal priority of defence areas.

Missile battery:

I have implemented single missile battery(called it “gun”), which is located in the middle of the ground. Aiming is performed with a crosshair and firing is controlled via mouse left-click. Whenever missile is launched, it will fly to the destination are with constant speed while not affected by the gravity (since I did not think that it would radically improve player experience, in my opinion the game is fun enough to play and adding gravity would just harden the task of the player). While missile is in motion collision rules do not apply, however when the particle starts exploding-all the colliding particles are affected by the blast. It is essential to mention, that I have used slightly modified version of Contact and ContactResolver from the lecture examples. Because of that and due to the fact that exploding missile stores and uses initial velocity as its impulse to resolve contacts, my explosions behave more like “vortexes” or singularities. For example, if an explosion collides

with a particle below the centre of explosion and the particle is moving down, it would most likely bounce up instead of accelerate down.

```
public void shoot() {
    if (expand()) {
        impulse = velocity.copy();
    } else
        position.add(velocity);
}

public void displayMissile() {
    if (explode)
        parent.fill(parent.color( v1: 255, v2: 140, v3: 0));
    else
        parent.fill(parent.color( v1: 128, v2: 128, v3: 0));
    parent.ellipse(gun_position.x, gun_position.y, expl_size, expl_size);
    parent.noFill();
}

//
private boolean expand() {
    if (expl_size == EXPLOSION_COEF * DEFAULT_SIZE) {
        vanish = true;
        return true;
    }

    if (position.x > destination.x - DEFAULT_SIZE/2 && position.x < destination.x + DEFAULT_SIZE/2) {
        if (position.y > destination.y - DEFAULT_SIZE/2 && position.y < destination.y + DEFAULT_SIZE/2) {
            explode = !explode;
            expl_size += DEFAULT_SIZE/2;
            return true;
        }
    }
    return false;
```

Fig.5 Missile implementation

Waves:

My implementation of the waves is based on the idea that in fact there is just one single continuous game loop with a lot of variables that change slightly from wave to wave. I have assigned each particle, city and missile a value and whenever particle is destroyed it increases the amount of players' points. Points are used as currency in order to restore destroyed city (if there are any) and the price of restoration is deducted from total score. Each restoration increases the price of the next one. I have done it this way in order to stimulate the player to gather points yet limit the amount of times restoration can be done (eventually the price would be too high). In addition, I am increasing gravity,missile amount, particle amount and particle appearance rate per wave, so the player will eventually loose because there would be too many particles that move at high speed to handle. This way I do not need to worry about the amount of levels.

```
private void generateNextWave() {
    if (!rebuilt_city)
        player_score += bonus_score;
    wave_particles_amount += 5;
    wave_particles_counter = wave_particles_amount;
    wave_counter++;
    particle_frequency = (particle_frequency > 30) ? particle_frequency - 10 : particle_frequency;
    n_missiles = new Double( value: wave_particles_counter * 1.5f).intValue();
    particles = new ArrayList<>();
    missiles = new ArrayList<>();
    gravity = (gravity.y < 6.5f) ? new PVector( x: 0f,  y: gravity.y + 0.3f) : gravity;
    playing = true;
    rebuilt_city = false;
}
```

Fig.6 Next wave generator

Whenever the last city is destroyed, the game displays a specified message with players score and asks the player to either start a new game or quit the programme by pressing specified keys. This is the only case when the game loop stops and it is restored if new game was chosen.

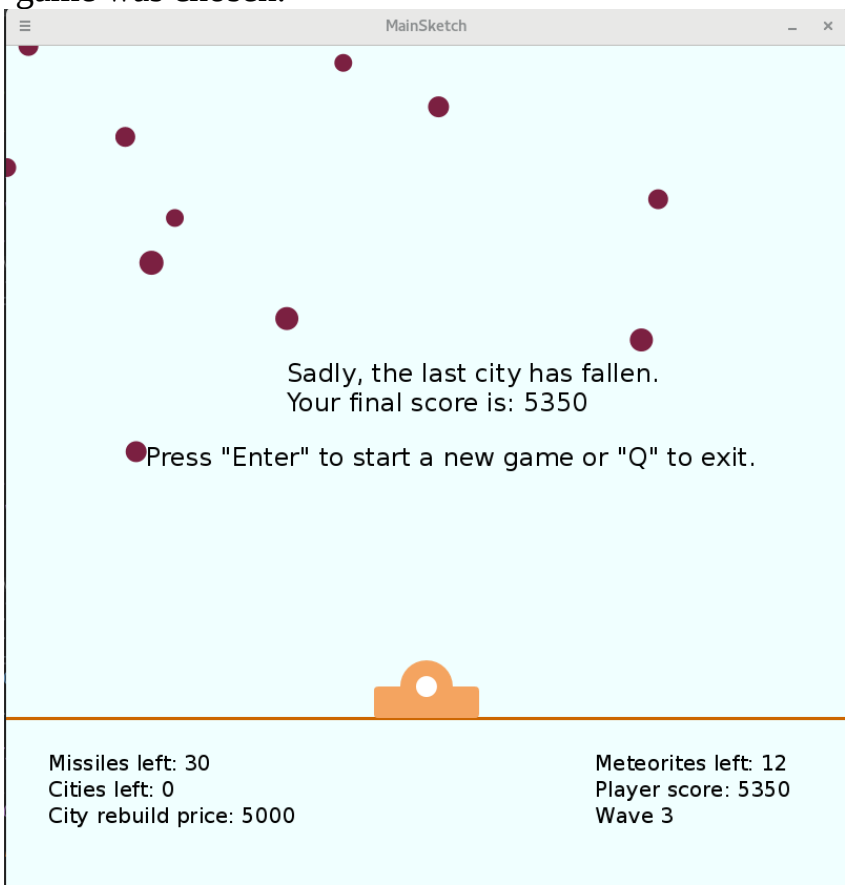


Fig.7 End game

Known issues (a.k.a. "it's not a bug, it's a feature"):

I have used basic collision detection resolver for particles and missiles (with adjustments and improvements) and have written my own particle-city collision detection logic, which is simply a check whether particle collides with any are of the city rectangle (by knowing leftmost point of it and size I could deduct the rest).

The only issue with particle-missile detection was specified earlier in missile sub-section. At first I was inclined to change it so it behaves as specified in basic requirements (force applied from the centre of explosion), however after playing the game for multiple times I found that behaviour funny and entertaining enough and after comparison with vortexes from one of my peers – I have decided to leave it as it is.

Another trick is done with score representation during inter-level "pause". Since total score is refreshed every draw() call and bonus points are calculated "on the run" in main loop as well, I didn't find the appropriate way of updating main score with bonus points while not breaking score count. This is why total score under the "ground" is updated with bonus points at the beginning of each wave, however they are still included when validating points for city restoration. This behaviour can be seen on Fig.8 and Fig.9.

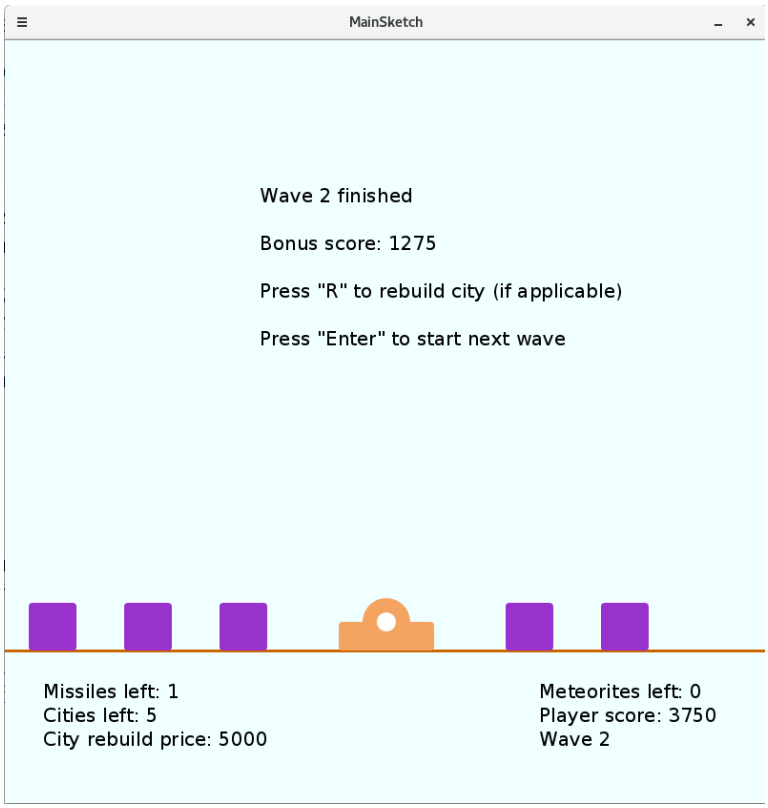


Fig.8 Before purchase

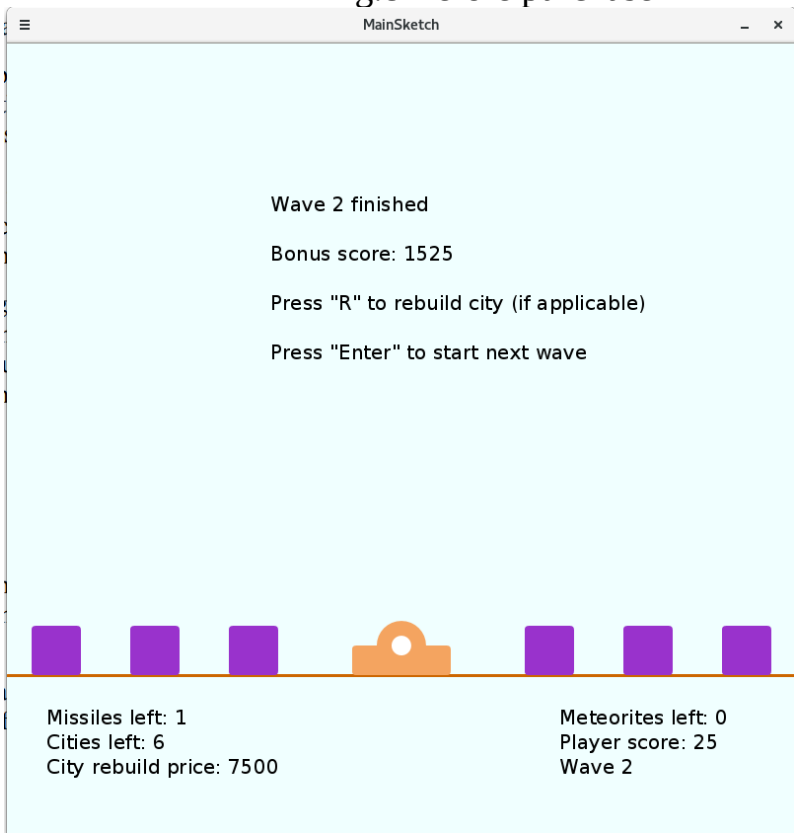


Fig.9 After purchase

Extensions:

I have implemented rendering of the flight of a fired missile, since it was relatively easy to do once I got my head around it (which took a while). I have also tried to implement the sound extension, however due to absence of required system-level libraries(tested on the lab machine) and uncertainty that such an issue would not raise during marking process – I have neglected the idea.
Evaluation:

Summary:

I have implemented a game according to all the basic requirements, as well as one additional extension. I have also tried to explain my decisions as clearly as possible in the report while not providing too much of a detail. The project was fun and interesting to work with and I have enjoyed playing the game once I have finished coding. If I were to publish it, I would probably implement correct explosion collision (however would leave my current one as “fun” mode”) and fix the score calculator issue. Apart from that, I think the game is decent enough minimalistic time-killer.