SAPIENZA
UNIVERSITÀ DI ROMA

Master of Science in Engineering
in Computer Science

a.y. 2018-19

INTERACTIVE GRAPHICS

Project Presentation – Bunker Buster

Presented by:

Lorenzo Altamura
MAT. 1538468
Paolo Mastrobuono Battisti
MAT. 1086895

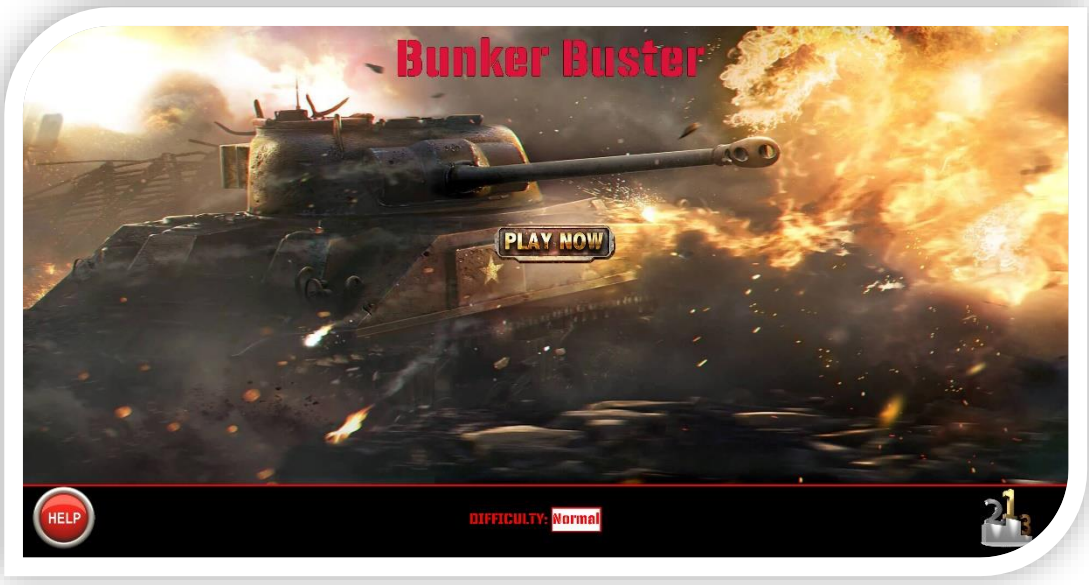
Submitted to:

Prof. Marco Schaerf

Index

Introduction

Bunker Buster is the title of a small game developed by the two candidates, the project was made basically by using JavaScript for the logic part and HTML plus CSS for the front end.



We made use of a series of external libraries (such as Three JS), tools, 3D models, pictures, songs and audio effects for which we don't own the rights since the project has no plans for commercial use.

The game allows the player to interact with a 3D model of a tank and score points by accomplish various achievements we will explain later.

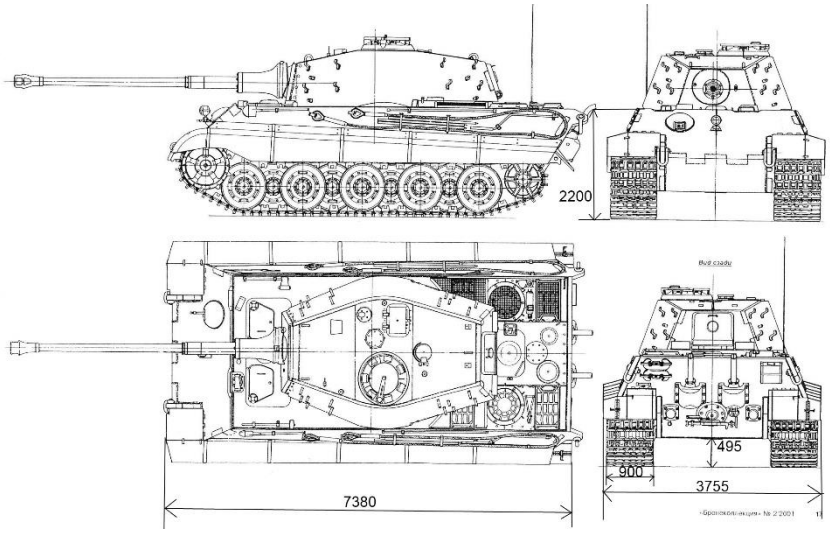
The game also offers various basic functions such as difficulty choice, username insertion and chart table.

This presentation will examine in detail various aspects of the game, starting from the inspiration for the game design, code samples and in-game screens for a better explanation.

We will write the sources of external materials (i.e. 3D models or libraries) when needed.

Part ONE: game design

The game is a top-down perspective shooter in which the player controls a tank. In particular, we've found a good looking and free-to-use model of a Tiger II (pz.Kpfw.VI Ausf.B), a German panzer from the World War Two.
The online platform where we've found the tank and various other free 3D models is called **clara.io**.



The goal of the game is to survive from the fire coming from five turrets situated in fixed spots all over the map and destroy them by shooting.

Every turret has his own health bar showing three different colors (green, yellow and red) marking his integrity state.

Time is a factor that effects the final score.

The player also has the possibility to get some power-up cubes around the map in order to get some useful temporary upgrades for 10 seconds. In particular, we have two boosters (**speed-up**, with a lightning symbol and **fast-shooting**, with three shells marking the cube) and a **health-points recovery**.

The structure of the game takes free inspiration from **Mass Destruction**, a game published by BMG Interactive Entertainment for PC, PlayStation and Sega Saturn in 1997 (a screenshot of the game in the previous picture).
In particular, camera position, shooting system and tank movements mechanics are very similar.
One more piece of software we needed to borrow is the background soundtrack.

We wanted to recreate a '90ish atmosphere, possibly choosing a background music matching with the simple style of the game but still resulting enough catchy.

The background sound of the game is taken for the third stage of Panzer Dragoon, a game developed by Team Andromeda and published in 1995 by SEGA for Sega Saturn.

Part TWO: the game interface

The main menu is very intuitive and easy to use.
In the left side of the footer, the player can read the main instructions to play properly, while in the right side of the footer we can see the chart with the names of the players and the relative final score.

In the middle of the footer, there's a "select" html object useful to choose one of the three difficulty settings.
As difficulty raises, the turrets' projectiles deal more damage to the player and have less reload time.

A big button "play now" is placed to start the game.

Once the game is up, in the middle of the screen will appear an area with the game rendered in real time.
The HUD (heads-up display) will show useful info for the player, such as the game time, the score obtained so far, the life of the tank, and the state of the power-ups (with the remaining time of the perks) .

Two more buttons have been implemented.
First one is in the upper right corner of the game area and lets the player to mute all the sounds.

Second one is placed in the footer and is an option to pause and resume the game.
Once the game is paused, one can restart the battle by using the appropriate button that appears in the middle of the screen.

Part THREE: the external libraries

All the libraries are imported as scripts in the *index.html* file.

Orbit Controls

Orbit Controls.js is a library that allows the camera to orbit around the target. We used it to place the camera where we needed it to stay.

```
function update_camera(){
  camera.position.set(tank.position.x,CAMERA_HEIGHT,tank.position.z);
  controls.target.set(tank.position.x,0,tank.position.z);
}
```

Three.js

Three.js allows the creation of Graphical Processing Unit (GPU)-accelerated 3D animations using the JavaScript language as part of a website without relying on proprietary browser plugins (from Wiki).
Most of our project makes use of it.

JQuery.js

jQuery is a fast, small, and feature-rich JavaScript library. It makes things like HTML document traversal and manipulation, event handling, animation, and Ajax much simpler with an easy-to-use API that works across a multitude of browsers. With a combination of versatility and extensibility, jQuery has changed the way that millions of people write JavaScript. (JQuery documentation).

```
sceneWidth = $(game_scene_div).width();
sceneHeight = $(game_scene_div).height();
```

```
function onDocumentKeyDown(event) {
  keyCode = event.which;
  if (keyCode === 86 && p1fireRate !== rate && shot) {
    sound_reload.play();
    event.preventDefault();
    event.stopPropagation();
  }
  else if (keyCode === 32 && p1fireRate === rate && !shot) {
    event.preventDefault();
    event.stopPropagation();
    sound_reload.stop();
    tank_shoot();
    shot = true;
  }
}
```

Bootstrap

Bootstrap is an open source toolkit for developing with HTML, CSS, and JS. Quickly prototype your ideas or build your entire app with our Sass variables and mixins, responsive grid system, extensive prebuilt components, and powerful plugins built on jQuery (Bootstrap Documentation).
We've used Bootstrap for the Modals in the help and in the chart.

```
<div class="column" id="help">
  <a data-target="#theModal2" data-toggle="modal">
    <img id="HLP" src="img/HelpBTN.png"></a>
</div>
```

```
  <div class="modal fade text-center" id="theModal2">
    <div class="modal-dialog">
      <div class="modal-content">
        <div class="modal-body">
          <h2 class="text-center">ISTRUZIONI</h2>
          <div id="command_image_div_id">
            <img id="commands_id" src="img/TankCommands.jpg">
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```

THREEx.KeyboardState.js

threex.keyboardstate is a threex game extension for three.js which makes it easy to keep the current state of the keyboard.
It is possible to query it at any time. No need of an event. This is particularly convenient in loop driven case, like in 3D demos or games. The syntax of the keys has been copied from jquery keyboard plugin to ease configuration.
It can help you control the characters of your three.js games (from the creator's page in GitHub).

```
if (keyboard.pressed("left")) {
  Turret_2.rotateOnAxis(new THREE.Vector3(0, 0, 1),turretRotateAngle);
  viewfinder.rotateOnAxis(new THREE.Vector3(0, 0, 1),turretRotateAngle);
}
```

Stats.js

This class provides a simple info box that will help you monitor your code performance.

FPS Frames rendered in the last second. The higher the number the better.

MS Milliseconds needed to render a frame. The lower the number the better.

MB MBytes of allocated memory. (Run Chrome with --enable-precise-memory-info)

CUSTOM User-defined panel support.

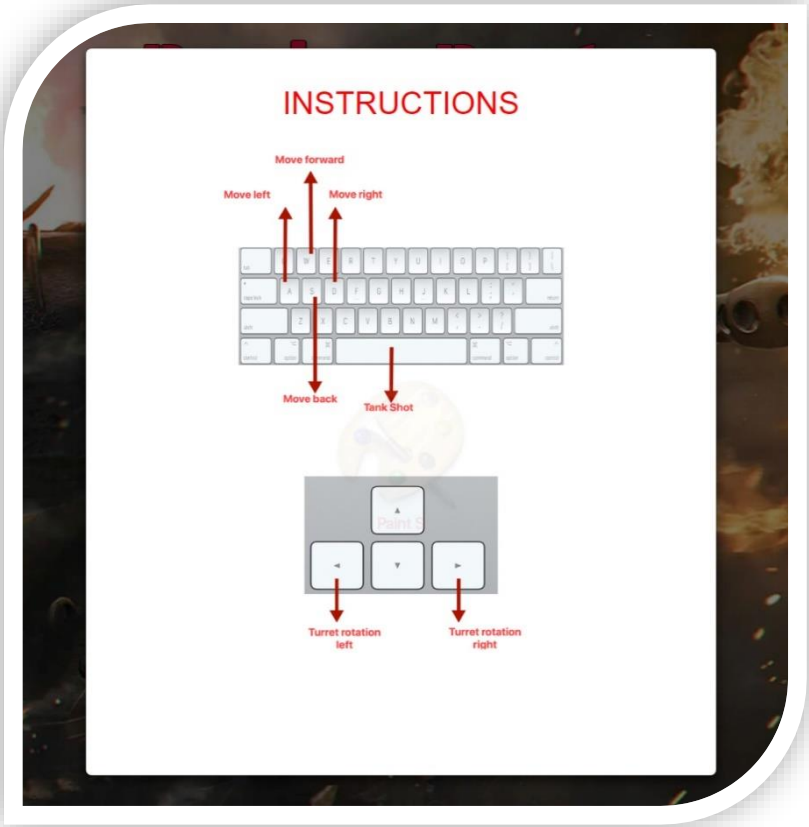
```
stats = new Stats();
stats.domElement.style.position = 'absolute';
stats.domElement.style.bottom = '0px';
```

Part four: the code

For the whole development of the project, we've used WebStorm, an integrated development environment (IDE) by Jetbrains, for Javascript (and HTML/CSS)

The Aesthetic (front end)

We made use of a combination of HTML and CSS.
The game is though to run on Google Chrome and Opera browsers.
We started working with the iframe but soon we realized that the limitations were too much and we preferred use a canvas loaded in a div contained in the body of the HTML.



For the *help* and for the *chart* we decided to use a bootstrap component called "modal".
The Modal plugin is a dialog box/popup window that is displayed on top of the current page.
Tip: Plugins can be included individually (using Bootstrap's individual "modal.js" file), or all at once (using "bootstrap.js" or "bootstrap.min.js") (W3School).
We made use of both implementations styles.

The logic

The logic is based on Three.js, a Javascript 3D library that makes the management of a 3D environment more easily manageable.

1) The commands

We disabled the base orbit controls commands in order to be able to use all the keyboard (since it makes many buttons unavailable).
We've used the Keyboard library and the functions used to capture the inserted commands (i.e. the keyboard's keydown).

```
function onDocumentKeyDown(event) {
  keyCode = event.which;

  if (keyCode === 86 && p1fireRate !== rate && shot) {
    sound_reload.play();
    event.preventDefault();
    event.stopPropagation();
  } […]
```

```
function move_tank(){
  if (keyboard.pressed("left")) {
    Turret_2.rotateOnAxis(new THREE.Vector3(0, 0, 1),
turretRotateAngle);
    viewfinder.rotateOnAxis(new THREE.Vector3(0, 0, 1),
turretRotateAngle);
  }
  if (keyboard.pressed("right")){    […]
```

2) The camera

Camera is fixed over the tank in "top-down" perspective.
It follows the tanks as it moves.

The camera is managed trough the Orbit Controls library.

```
function update_camera(){
  camera.position.set(tank.position.x,CAMERA_HEIGHT,tank.position.z);
  controls.target.set(tank.position.x,0,tank.position.z);
}
```

3) The interactions

A crucial part of the project was to manage the interactions between the objects in the canvas.

Since the game is a shooter, we had to deal with collisions between projectiles and enemy turrets.

We checked intersection between shells and turrets as it follows:

```
function shoot_controls() {

  for( let i=0;i<bullets.length;i++) {
    for (let z = 0; z < cannons.length; z++) {
      if (bullets[i].visible === true
          && cannons[z].visible
            && bullets[i].position.x >= cannons[z].position.x - 10
              && bullets[i].position.x <= cannons[z].position.x + 10
                && bullets[i].position.z >= cannons[z].position.z - 10
                  && bullets[i].position.z <= cannons[z].position.z + 10)
                   […]
```

We also had to deal with collisions between player's tank and enemy turrets.

We managed it in the following way:

```
[…]for (let i = 0; i < cann_positions.length; i++) {
  if (cannons[i].visible === true
      && clone.position.x >= cann_positions[i][0]-5
        && clone.position.x <= cann_positions[i][0]+5
          && clone.position.z >= cann_positions[i][2]-5
            && clone.position.z <= cann_positions[i][2]+5) {
    return false;
  }[…]
```

4) Meshes insertion

Most of our models are external and imported as *.json* files by using a Json loader. Here's a sample:

```
function addTank(){
  loader = new THREE.ObjectLoader();
  loader.load("models/tank/tank.json",
    function (obj){
      tank = obj;
      NUM_LOADED++;

      tank.scale.set(4.5, 4.5, 4.5);
      scene.add(tank);

      Body_1 = scene.getObjectByName('Body_1');
      Body_2 = scene.getObjectByName('Body_2');
      Track = scene.getObjectByName('Track');
      Turret = scene.getObjectByName('Turret');
      Turret_2 = scene.getObjectByName('Turret_2');

    });
}
```

Other meshes in our project are generated through Three.js itself and his *THREE.MeshBasicMaterial* function:

```
function add_speedcubes(nCubes){
  const Material = new THREE.MeshBasicMaterial( { map: speed_cube_texture
} );
  const Geometry = new THREE.CubeGeometry(30, 30, 30);
  speedcube = new THREE.Mesh(Geometry,Material);

  for(let i = 0; i<nCubes; i++){
    speedcubes[i] = speedcube.clone();
    speedcubes[i].position.set(generate_random(),8,generate_random());
    speedcubes[i].scale.set(0.4,0.4,0.4);
    speedcubes[i].visible=false;
    scene.add(speedcubes[i]);
  }
}
```

5) The game over / restart game

Once the player achieves the goal, a dedicated function will manage a series of actions, such as chart insertion and game restart.

```
function game_over(par) {
  reset_global_vars();

  SCORE -= curTime.toFixed(2);
  if(SCORE < 0){
    SCORE = 0;
  }
  let temp_div = document.getElementById("game_over_div_id");
  let text_to_change = temp_div.childNodes[0];

  if (par === 0) {
    document.getElementById('score').innerHTML = "Score: " + Math.floor(SCORE);
    text_to_change.nodeValue = 'GAME OVER';
    document.getElementById("game_over_div_id").style.display = "block";
    explosion.play();
  }
  else if (par === 1) {
    SCORE += 100;
    document.getElementById('score').innerHTML = "Score: " + Math.floor(SCORE);
    text_to_change.nodeValue = 'VICTORY';
    document.getElementById("game_over_div_id").style.display = "block";
  }

  let id = requestAnimationFrame(animate);
  cancelAnimationFrame(id);
  mute_unmute_game(2);
  save_high_score(SCORE);
}
```

Part five: the code: Project Requirements

Hierarchical models

The tank is composed by various parts, such as body, track and turret.
For example the turret is hierarchically linked to the hull as it can both rotate independently and translate in the space together with the hull.

```
"object": {
 "children": [{
[…]
{
      "name": "Turret",
      "uuid": "C1BAA599-26A7-3D14-8FF3-4A89B7E98201",
      "matrix": [-1,0,0,0,0,0,1,0,0,1,0,0,0,0,0,1],
      "visible": true,
      "type": "Mesh",
      "material": "C1BAA599-26A7-3D14-8FF3-4A89B7E98201",
      "castShadow": true,
      "receiveShadow": true,
      "geometry": "94B2F238-D0BE-3442-A73A-C6BD0F0C1ED4"
    }]
[…]
```

Lights and Textures

In the real time rendered game, we added at least a couple of textures: one for the terrain (a simple pattern repeated many time), and the texture needed to shape the power-up cubes.
Almost every single externally implemented mesh has his own texture file.

Game is lighted by a THREE.PointLight working as the sun.
If the player's health goes below a certain margin, light turns red until a health cube is picked to restore life.

```
light = new THREE.PointLight( 0xffffff, 1.5,0 ,2 );
light.position.set(200,1000,200);
light.shadowCameraVisible = true;
  light.castShadow = true;
  light.shadow.mapSize.width = 1024;
  light.shadow.mapSize.height = 1024;
  light.shadow.camera.near = 0.5;
  light.shadow.camera.far = 10000 ;
```

User interaction

The project is essentially a game, so everything goes around interaction.
The tank is under the control of the player and it can both move around the map and shoot to the enemy turrets.

Animations

This point is strictly linked to the first: as the player controls a tank composed by various parts organized in a hierarchical structure, we exploited the dependency between turret and hull to allow the player to:

-Rotate the hull only, turret included

-Rotate the turret only, hull excluded

-Rotate both the hull and the turret, even in opposite directions

Anytime the tank moves forward or backwards, the turret translates with it.

Finally, all the enemy turrets are animated as they always look at player's tank.