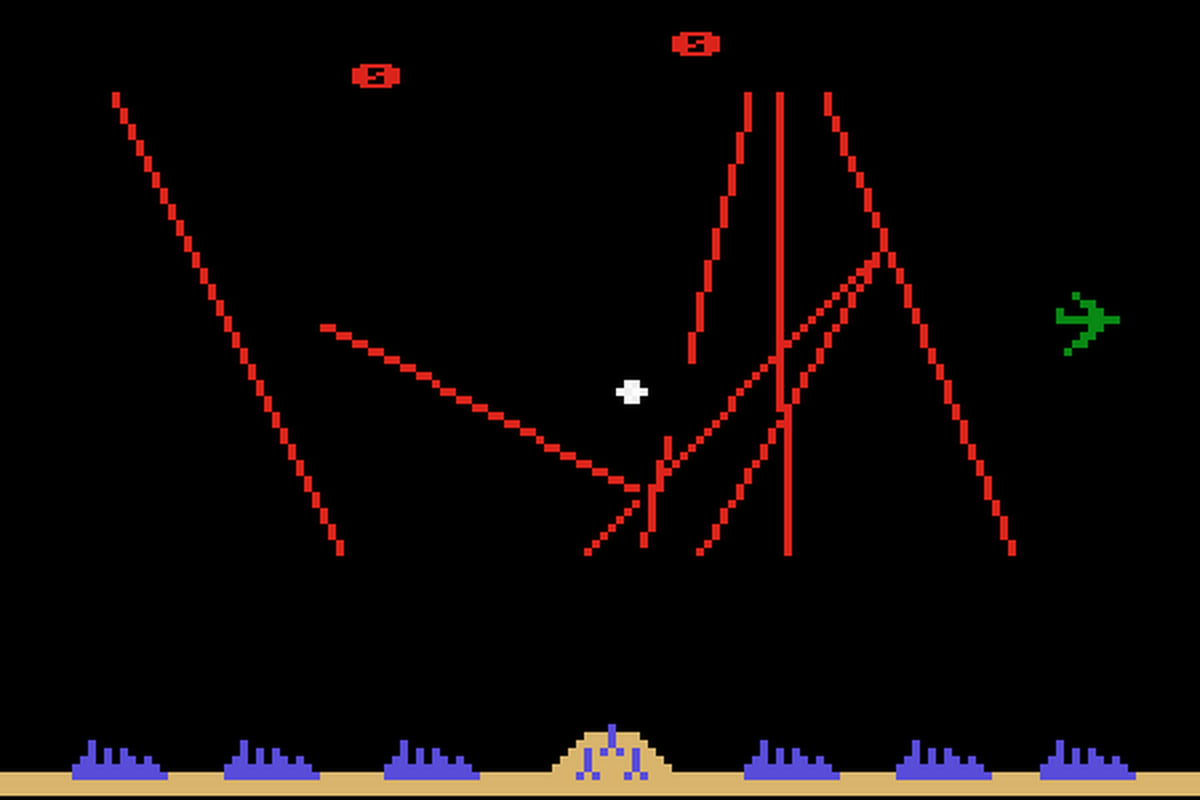
**ETGG3801/3802: “The Game” Design Statement / Scoring Rubric Updated: 8/26/2019**

*The purpose of this document is to specify the design goals, mainly from the user’s perspective. It doesn’t say anything about* how *these goals will be met (in general). We’ll repeat this lab, once for each of the 3 engines (Unity, Unreal, and ssuge in ETGG3802*

**Elevator Pitch:**



Blaster Master (Sunsoft, 1988)



Missile Command (Atari 1980)

*A combination of elements from Missile Command and Blaster Master.*

**Gameplay Features / Descriptions:**

* Very important: I want you to use *only* those assets I provide (so we’ll be able to do a more apples-to-apples comparison of the engines. You are *not* to use any plugins, asset packs, etc. – everything should be done in the “vanilla” engine.
* If you’re not sure of how something should be implemented, **ask** (preferably in class so we can discuss it as a group)

1. Gameplay is on a fixed-size plane of “voxels” that is larger than the player can see at one time. Make it procedurally generated with a gentle 2d sin-wave.
2. Periodically, missiles come from space. The player tries to stop them. If the player is unsuccessful in stopping the missile, a part of the play field is removed (use a gravity-falloff + disappear effect), making the game more difficult.
3. If a missile is destroyed, it drops money, which can be picked up by the player.
4. There are several turrets located on the playfield. If the player reaches them, they can expend a little money to fire a defensive missile. Here the player can fire at an incoming missile by launching their own rockets which fly to a designated point and explode, destroying any enemy missiles within range. Each turret has a limited cone in which it can fire.
5. Player can also spend a moderate amount of money to make a turret go into auto-fire mode for a while.
6. Player can enter the base to spend some money to repair part of the ground.
7. The base has a shield. Each missile hit depletes this shield. If it reaches 0, game over.
8. Particle Effects: smoke trail on all missiles, explosion when a missile hits something.
9. Simple mini-map showing incoming missiles, player location, turret + base locations and fallen money.
10. 3D sound effects.
11. HUD displays money amount and base shield level.
12. Simple titles screen that goes to a simple menu: new / save / load / quit (this menu will come up in-game when escape / start is pressed)

* Minor bonus points for going above and beyond.

**Major Bonus Features[[1]](#footnote-1)**

* Minecraft-style local networking (ask if you want an explanation) – each player has their own distinctly colored avatar. The group shares money and base-health.
* VR mode: one of these two approaches:
  + If Networking is also in place, the VR player could be a “dungeon-master”, with a “god-mode” view and the ability to control the missiles being fired at the other players.
  + If no networking, have some kind of click + pathfinding to control the player.

**Scoring (out of 300 points)**

1. (1 hour) Spawn the grid of voxels. Make the amount and voxel mesh configurable in-engine. The location should be centered around a “spawner” object (this might be where you attach the spawning code)
2. (2 hours) Spawn the base on the middle cell on the map (pick one of the 2-4 central cells if you have even world width / height). Spawn x launchers in a ring around the base (but still located directly on top of one voxel). Make x configurable by the user. Make each of these objects a child of the block they lie upon so when the block falls, so does the object.
3. (6 hours) Make the player object. I \*don’t\* want you to use any kind of built-in player controllers – do it manually. The controls I want are:
   * (mouse + keyboard): mouse (or left/right) rotates the view around the player. Lock the mouse. W/S moves forward / backward and A/D moves right. Space jumps.
   * (gamepad): right-analog rotates camera, left-analog moves forward/back and strafe. A jumps.

Make the player physics-controlled (with gravity). Make the player’s walk and idle animations play.

When the player touches a launcher after a short camera-lerp, hide the player model (and stop things like jumping). In this mode, use these controls:

* + (mouse + keyboard): mouse (or left/right) still rotates the view. Q/E raises or lowers a firing plane mesh. WASD moves a target point within a circle (higher plane = wider radius, defining a cone). Space fires a missile towards from the launcher to the target point.
  + (gamepad): right-analog rotates view. left-analog positions the target. A fires.

1. (2 hours) Missile Logic. When an enemy rocket hits a voxel, that voxel and those within a fixed radius fall. Now the player has to jump over gaps. When a player object gets within n units from an enemy missile, destroy both.
2. (2 hours) Turret logic: When the player spends money on the auto-fire mechanism, find a missile closest to the vertical line coming up from the turret and fire at it. Try to (crudely) predict where it will be and aim there.
3. (4 hours) Base logic: when shields reach zero, base falls and game-over. When player spends money to heal the ground, use something similar to the turret firing mode where the player drags a target over the area to heal. Resurrect any dead voxels within a fixed range of that point.
4. (4 hours) GUI: Show health and base-hp on-screen. When the player is in turret firing or base-mode show a little text-gui that has key/button presses for auto-fire, heal-mode. Use the last input device to determine your text (e.g. if keyboard was used last, draw “[Space] to auto-fire”, but if gamepad was used last, draw “[A] to auto-fire”
5. (2 points) Other polish items (particle effects, 3d sound, etc.)
6. (2 hours) Menu
7. (1 hour) Build an executable.
8. I’ll have you demo your project in-class. I might ask you to submit all (or part of) your project on blackboard as well.

1. Don’t attempt these unless you are *very* confident in your skills. You’ll get many more points by doing the regular parts of the lab. [↑](#footnote-ref-1)