3 Interim report

3.1 Progress Report

We are quite happy with our progress so far, having achieved most of our “Low target” (level 2) priorities well before their deadline, such as:

- A moving camera (ab)
- Active Shield System (ab)

From our “desirable target” (level 3), we have been mainly focusing on

- Destructibility of planets (dz)
- Sound (rms)
- Random map generation (rms)

3.1.1 Gameplay

At the current state, up to four players can play our game by navigating their spaceship through a two-dimensional playing field. By shooting lasers or rockets, they can damage the opposing spaceships and destroy the planets randomly distributed on the playing field. The planets build up a gravity field, influencing most objects currently in the scene. Each player has a shield equipped, protecting him from damage for a short amount of time. The shield mechanics work as planned: It depletes quickly while in use, and recharges slowly otherwise. By picking up items on the playing field, players can gain a limited amount of weapon power-ups or repair packs, which include at the moment

- ○ 30% health repair packs,
- ○ Rapid-fire laser enhancement,
- ○ Fan-laser enhancement,
- ○ Rockets, which can be triggered remotely, causing damage in a large area.

For now, every player takes damage by being hit from a shot, being caught in an explosion or by colliding with an obstacle at high velocity. When the player’s “health” is used up, the spaceship explodes and respawns after a few seconds. In near future, we will decide on the game modes to implement.

From feedback we got in early playtesting, it became clear that two of our originally three control schemes clearly separated players into two crowds of “casual players” and “classic players”.

For “classic players”, ships control as in many old-school space games such as Asteroid: Players indicate a movement direction with the left thumbstick, and accelerate in that direction by pressing the A button. This is harder to control, but allows for complicated maneuvers such as shooting against the current movement direction.

For “casual players”, the left thumbstick controls both the direction of movement and the acceleration in that direction. This makes it harder to take turns without losing speed, but is easier to pick up by new players.

To accommodate everyone’s demands, we have included both these control schemes, as well as a third scheme, which is modelled after early-3D era (e.g. PlayStation 1) “turret” controls: The left thumbstick now controls rotation relative to the ship’s current rotation (instead of relative to the global axes). This means that by setting the left thumbstick to 9 o’clock, the ship will continuously rotate to the left.

Players can choose their preference before a game round is started. Per default, the “casual” scheme is selected.

Another important gameplay mechanic currently in discussion is the players’ weapon/item inventory. One decision is to give the player a primary weapon (the laser and its variations) plus a set of secondary items (such as rockets). These two categories are assigned different controls. For the secondary items we first intended to use a browsable list (you could switch to the next item with a button press). However, we are currently leaning towards a more “just use the last thing you picked up” oriented mechanic. We hope that this will prevent additional complexity during the heat of battle.

3.1.2 Physical behavior

Each scene objects follow modified physical rules. As such, although the setting is in space, objects are bound to friction, explosions occur and sound is created.

A planet is represented as a two-dimensional polygon in the plane. Other scene objects are bounded by circles for collision detection and response, which are checked against other objects in a continuous way; other objects such as laser shots are simple points and are therefore best represented by line segments indicating their trajectory since the last frame. Effort has been made to ensure that objects never arrive at the inside of planets, and that collision response follows the player’s intuition. This includes realistic bumping from collision partners with certain energy loss, and taking damage when colliding with high speed. With friction introduced, a player is even able to land on a planet.

Destruction of planets is subtracting a spherical explosion shape from a planet’s shape. This is implemented by splitting a planet’s boundary at intersection points, and filling the resulting holes with vertices sampled around the emerging crater. We took care to pre-allocate the unknown amount of vertices used, such that heap-allocations during runtime can be minimized. The resulting polygon is then re-triangulated for rendering using the ear-cutting algorithm. If a planet splits up into two or more separated components, we decided to let the split parts stay as they are, which means that they do not follow the gravity field’s influence. We plan to justify this graphically by drawing the ‘back half’ of a planet regardless of the destruction, which is behind the plane of the playing field. Destruction of planets also causes the gravity field to change due to the loss of mass.

3.1.3 Graphics

Spaceships are currently drawn as 3D models in the player’s color with a simple Phong shader. Their engine emits a plasma-like trail made from particles. Laser shots which players emit are drawn as point sprites in the same color as the shooting player. Rockets, pieces of chunk and collectable items and other planned 3D models are currently drawn using spherical placeholders.

Planets (and their remains) are flat triangulations of their boundaries, but are drawn as 3D textured spheres using a pixel shader. The spherical shape still remains in case of destruction.

For item pick-ups or chunks (space debris) flying around, there are currently placeholders indicating their locations. For debug purposes, the gravity field is currently drawn as a vector field.

The camera ensures that all players are displayed within the scene boundaries, moving in a smooth manner. Around the playing field, an exclusion zone is drawn with a striped background, in which players will (later in development) be destroyed if present for too long. The camera only displays this exclusion zone up to a certain distance from the playing field.

Particle effects are used for explosions and for dust dispersed from objects colliding with planets. Damage on a ship is shown by using a smoke trail for light damage, and increasingly furious flames for higher damage, until the ship explodes.

3.1.4 Obstacles & Challenges

While working on the automatically adjusted camera, we had to simultaneously decide the way in which the boundaries of the playing field had to be handled. Due to the fact that the bounding box in which all players' spaceships are contained might possibly show more than the whole playing field, our boundaries must extend to the maximum are coverable by the camera.

In quite the same vein, we had to decide on a default aspect ratio for our screen space: As the majority of resolutions supported by the Xbox 360, as well as most TVs and computer monitors, are in Widescreen (i.e. 16:9), we decided to make this aspect radio our default. We still take care to support taller ratios such as 4:3, especially for VGA (640x480), also a resolution available on Xbox 360.

During development so far, we were both careful and lucky enough that our memory-friendly design has shown no performance problems, thanks to many design decisions that avoid run-time memory (de)allocations:
We use compact and reusable data structures instead of classes where possible.
We make use of ref- and out-parameters where sensible to return multiple values while avoiding implicit copy operations.
For resource management, we make heavy use of helper data structures such as object pools and array pre-allocators, which can be created before the start of a game round.
We designed our class structures in such a way that we reached a compromise between clarity of design (polymorphism for reusability) and run-time performance (avoidance of multiple virtual calls at once). This is especially visible in our collision detection codebase, which has to exhibit excellent performance at all times, as it is executed many times every

frame.

3.2 Screenshots

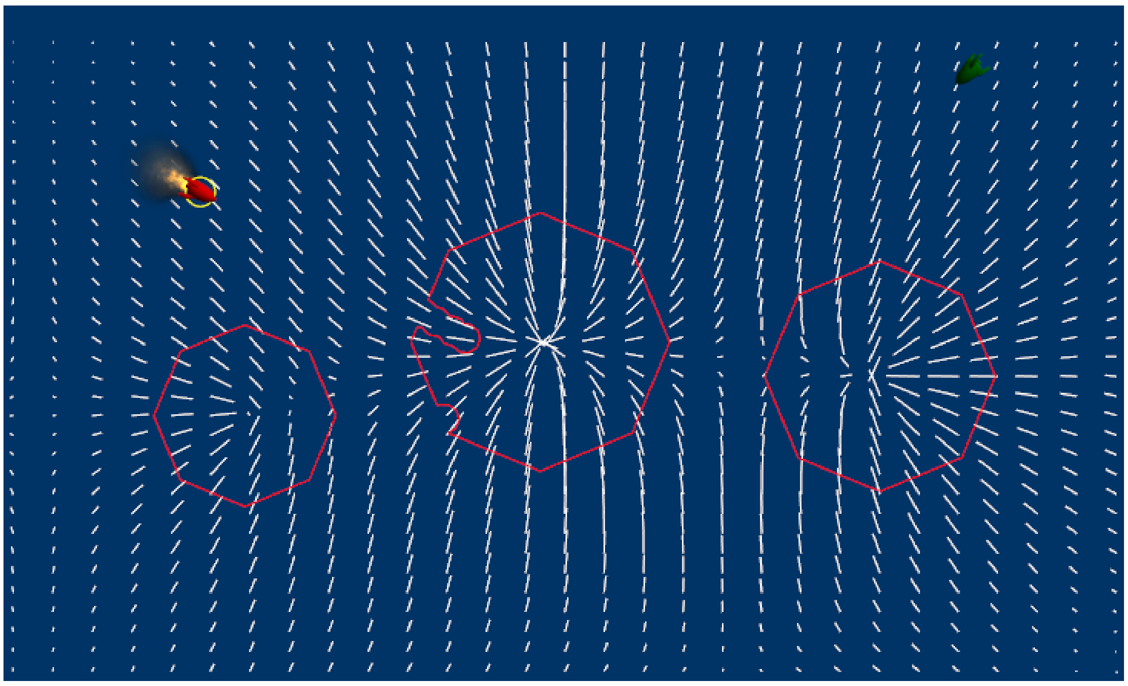


Fig. 3.1: Shows basic triangulation patterns of planets after a certain amount of explosions.

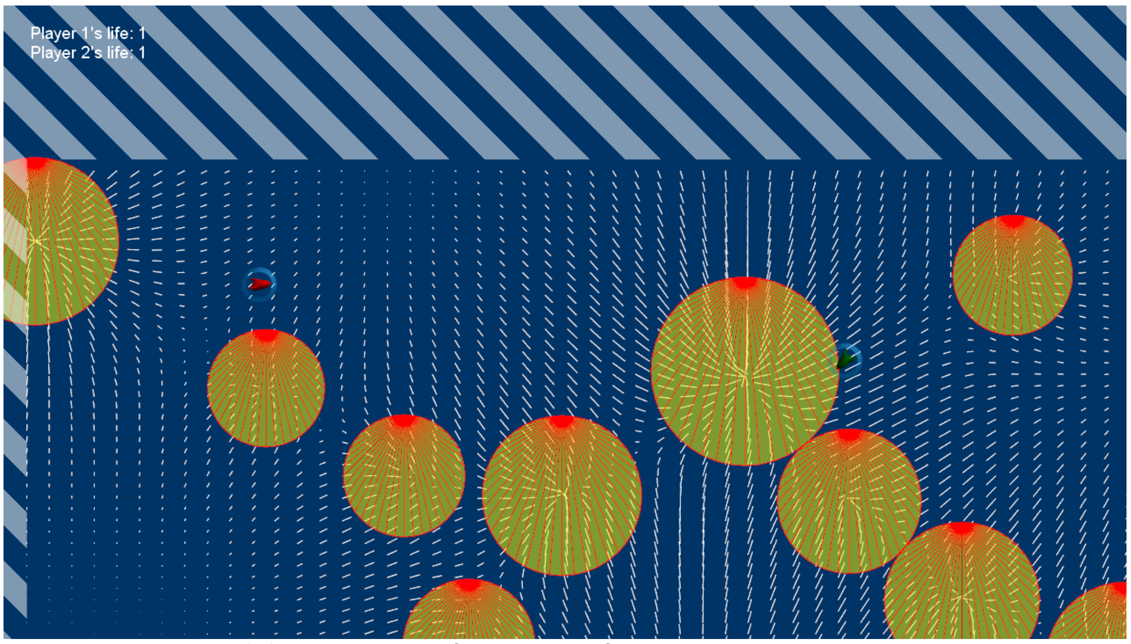


Fig. 3.2: Boundaries of the playing field are now rendered using warning stripes.

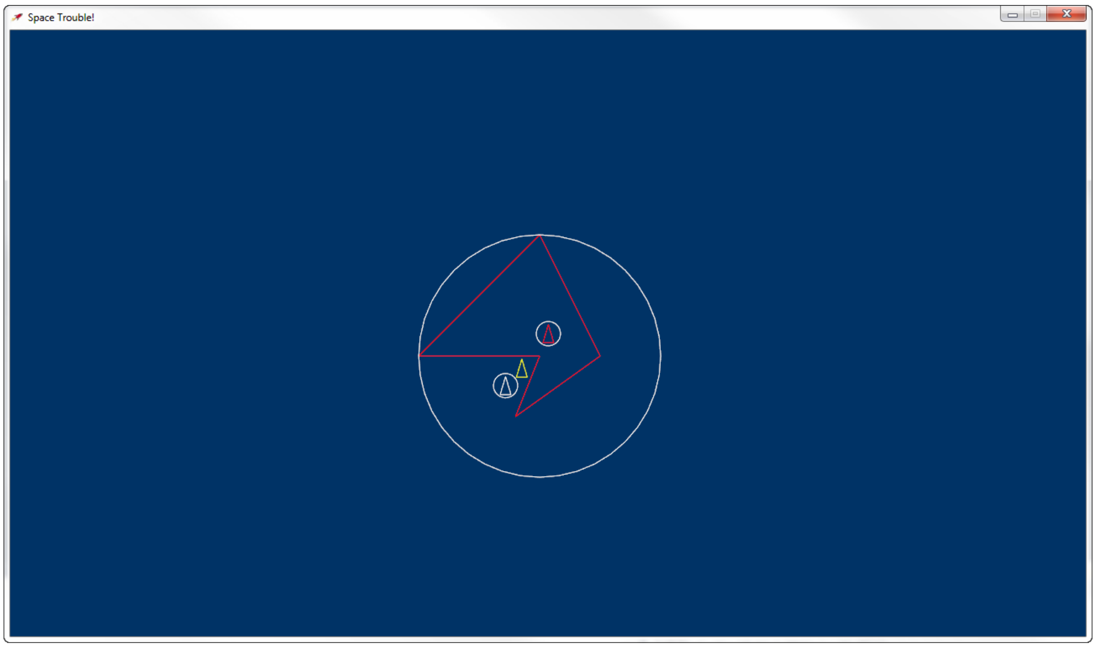


Fig. 3.3: Collision testbench: Spaceship positions from the previous and the current frame can be set. The position after iterative collision detection and response are shown in yellow.

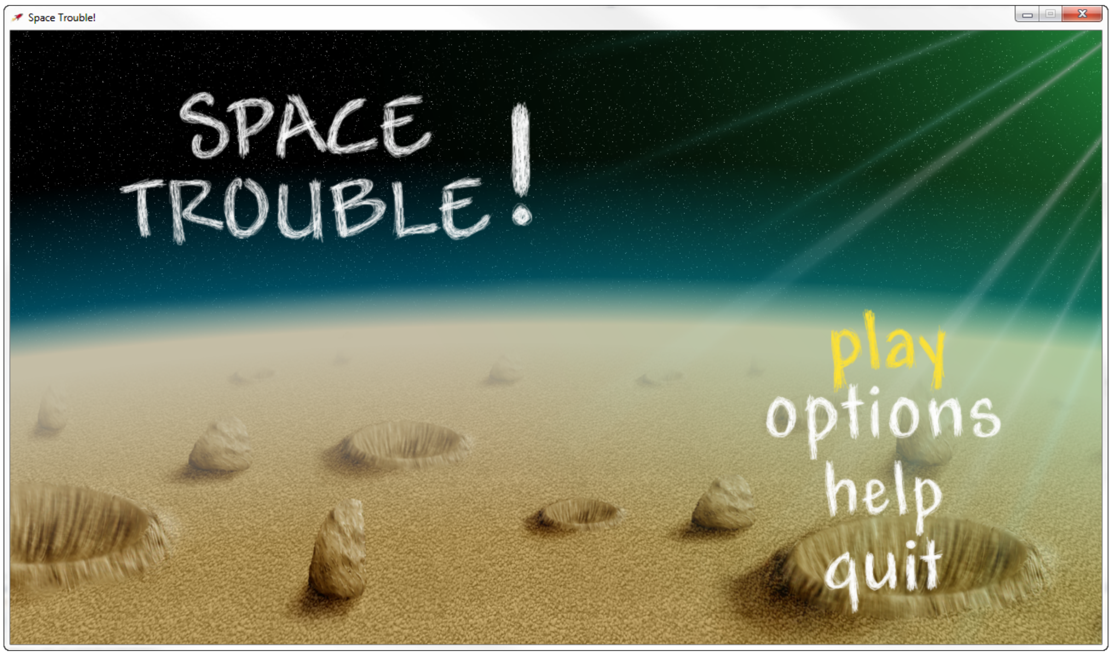
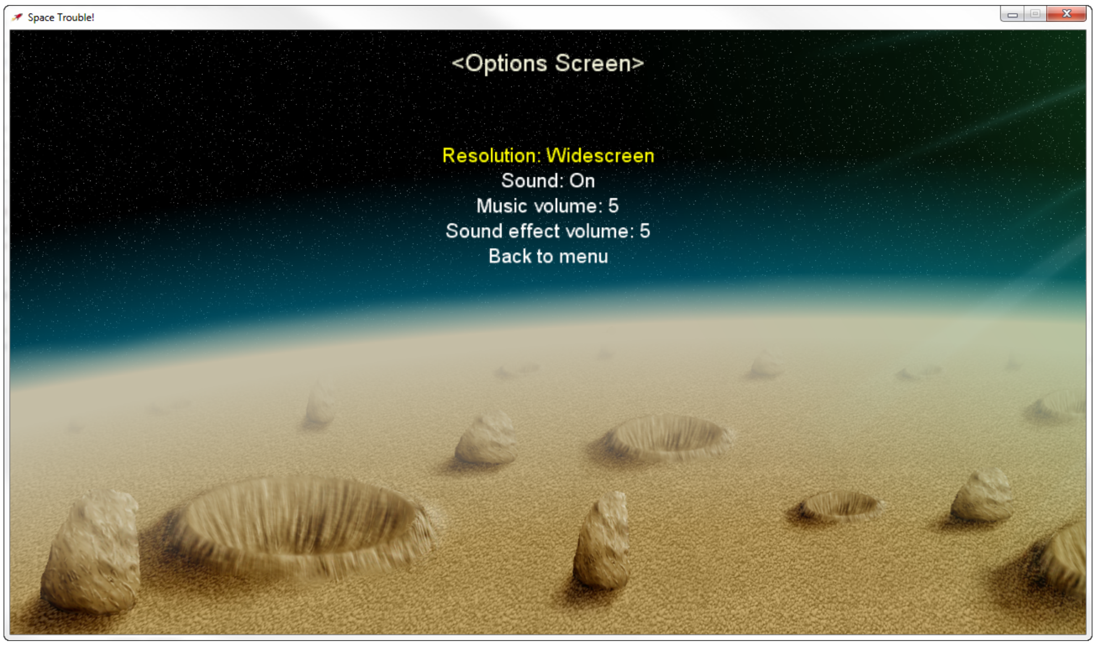


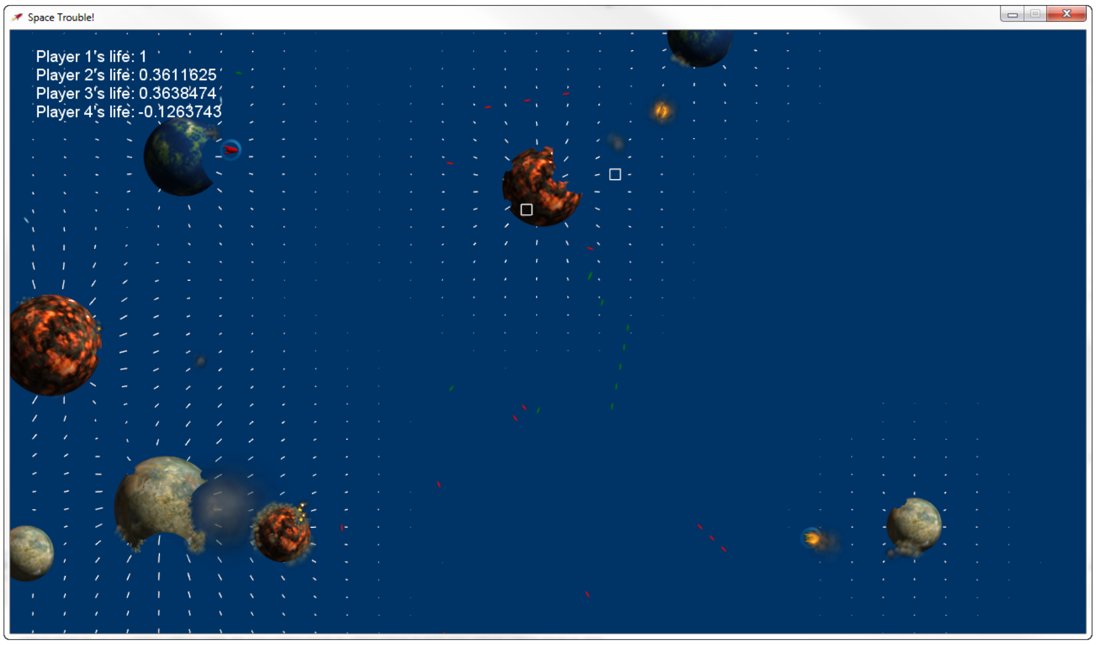
Fig. 3.4: The main menu.



<Options Screen>

Resolution: Widescreen
Sound: On
Music volume: 5
Sound effect volume: 5
Back to menu

Fig. 3.5: The current options menu.



Player 1's life: 1
Player 2's life: 0.3611625
Player 3's life: 0.3638474
Player 4's life: -0.1263743

Fig 3.6: A current screenshot of a gameplay state.

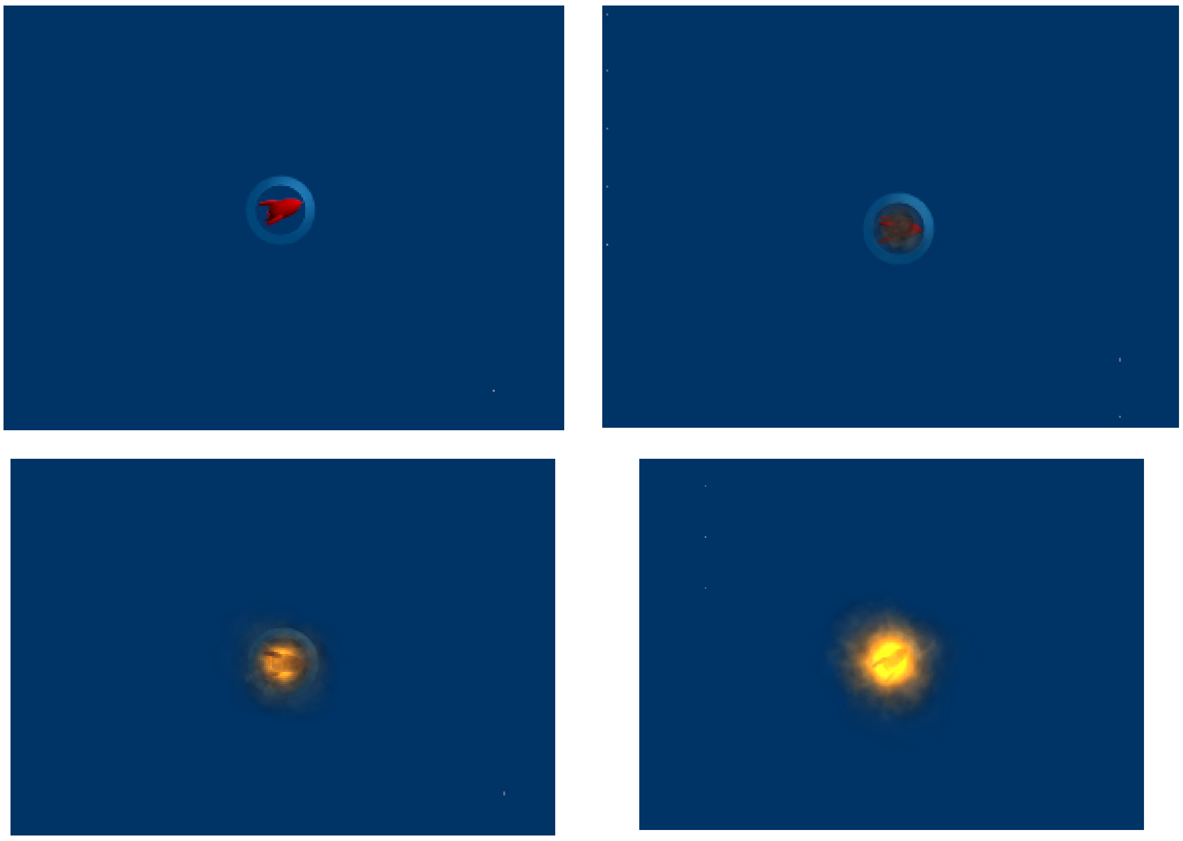


Fig 3.7: Spaceship damage visualization.

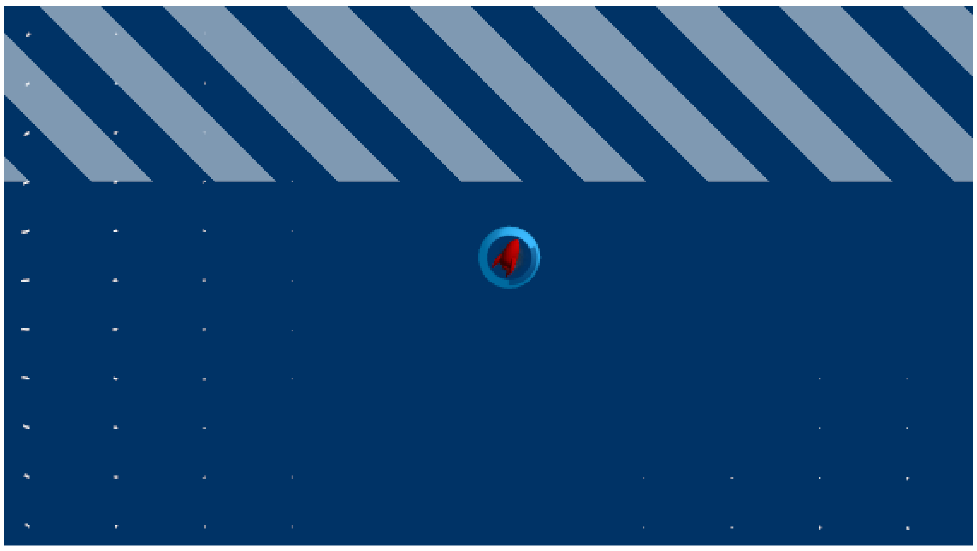


Fig 3.8: The shield visualisation showing an active, partially depleted shield.

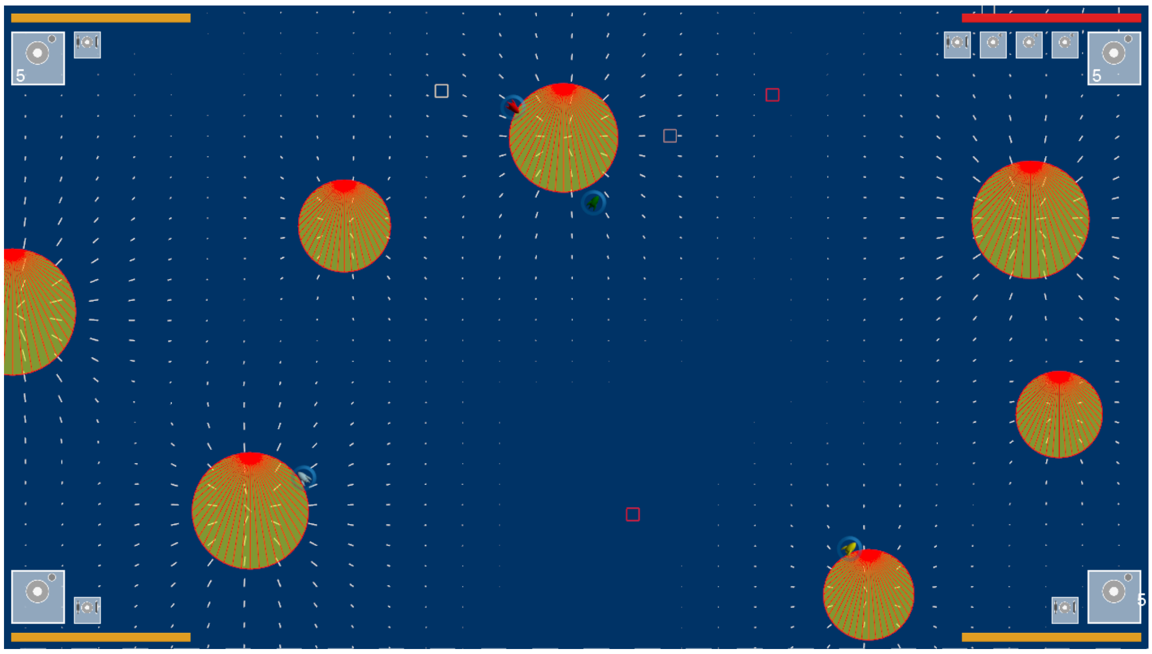


Fig 3.9: Current state of the UI branch.