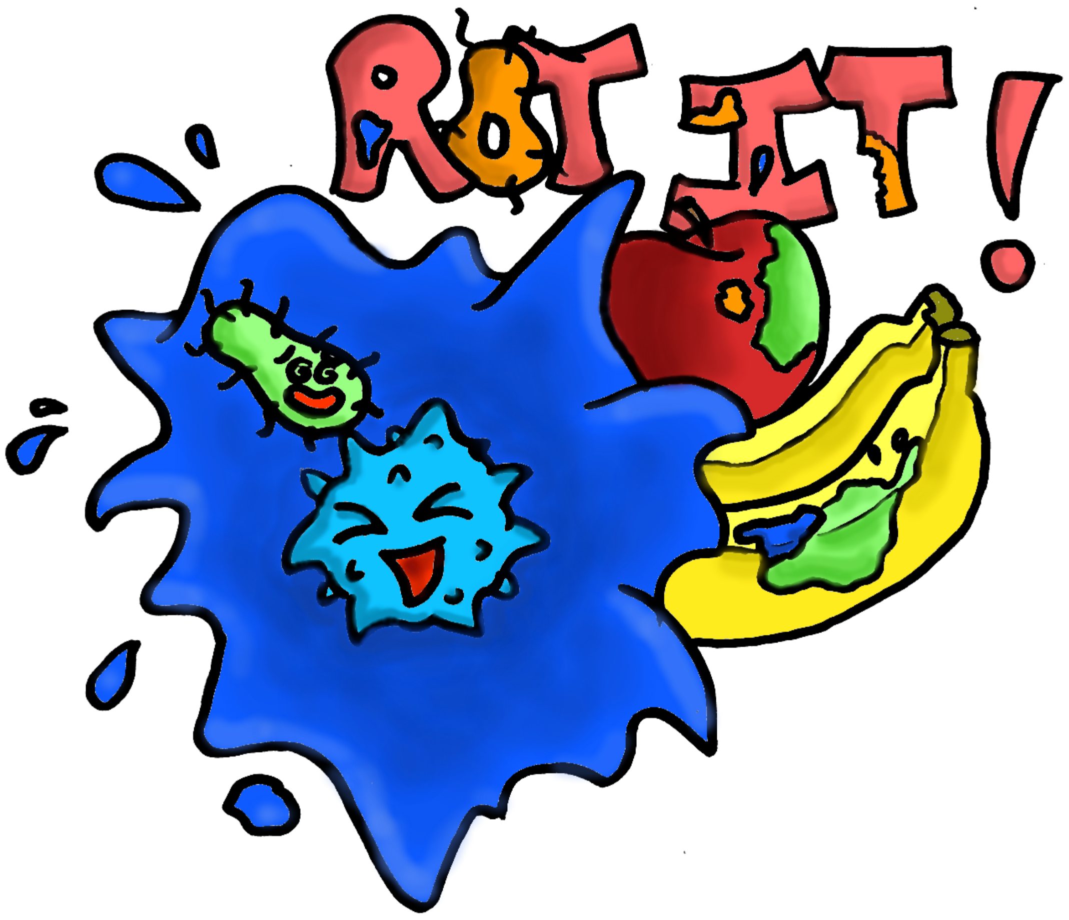


Have you ever seen perfectly shaped food not ever going bad ?
Have you ever wished that you could do something about it ?
Now is your chance! Pick your bacteria type and decay that food!
But be careful, other bacteria will try to sabotage you.
Battle for the right to decay the food in your own way!

Team 4
Ioana Pandele
Irene Baeza
Carlota Soler
Daniel Borges

Team 4: Ioana Pandele, Carlota Soler, Daniel Borges, Irene Baeza

Interim report

Task allocation

Done
In progress
To do

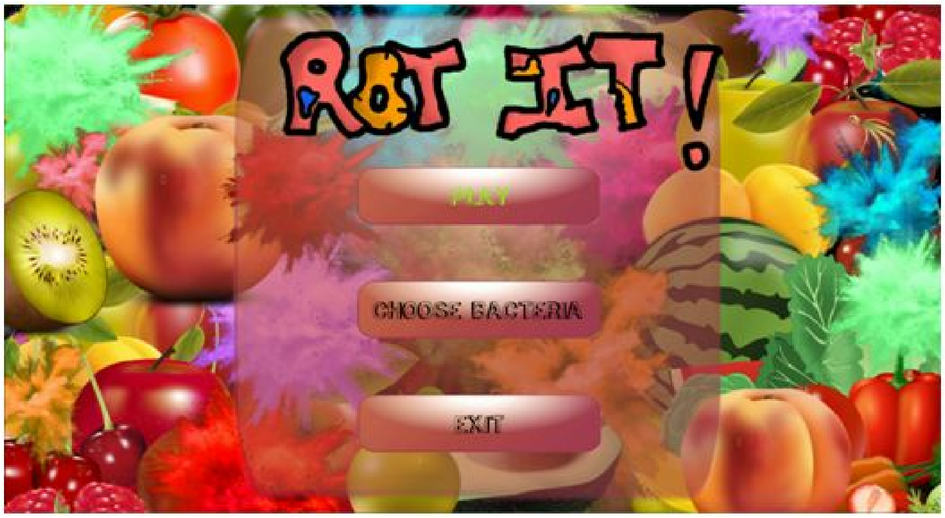
							Easter	1st play demo					Alpha release		
Task name	Time (hrs)	w1	w2	w3	w4	w5	w6	w7	w8	w9	w10	w11	w12	w13	w14
Assignments															
Game Proposal Draft	4		All												
Game Proposal Chapter	6			All											
Prototype Chapter	6				All										
Interim Report	10									All					
Alpha Release Chapter	10											All			
Playtest Chapter	6												All		
Conclusion Chapter	6														All
Deliverables															
Physical Prototype	16				All										
Alpha Release	-											All			
Trailer	-													All	
Miscellaneous															
Game Idea Presentation	2		All												
Final Game Idea/Prototype Presentation	2					All									
Monogame Setup	6				All										
Bug Fixing	-										All				
Playtesting	-											All			
Playtesting Presentation	-												All		
Public Presentation	6													All	
Functional Minimum															
Basic Modelling + Integration	8					Daniel									
User Controls Integration w/ movement	12					Irene									
Basic Explosion (stains)	8					Carlota									
Player score management	8						Carlota								
Collision Detection	16					Ioana									
Basic resource placement	8						Irene								
Low Target															
Player interactions	8							Carlota							
Explosion stains (shape and scoring)	20							Carlota							
Screen (Scores and Lives display)	8								Irene						
Handling Collisions with env.	20							Ioana							
Movement physics	6								Ioana						
Explosions (Particles System)	16							Irene							
Modelling upgrade and integration	20							Daniel							
Desirable Target															
Explosion integration	8									Irene					
Stain interaction with walls and particles	20									Carlota + Irene					
Explosion preview	8									Carlota					
Advanced skills (dashing)	20										Ioana				
Advanced resources	8											Carlota			
Animations integration	15										Irene				
Sounds	16									Daniel					
High Target															
Start Menu	16											Irene			
Special obstacles and resources	16											Carlota			
AI	12											Ioana			
Graphics improvement (animations)	15											Daniel			

Menu

We added a start Menu to our game. All players can move from one option to the other using the gamepad left thumb stick. The Menu consists of the following screens.

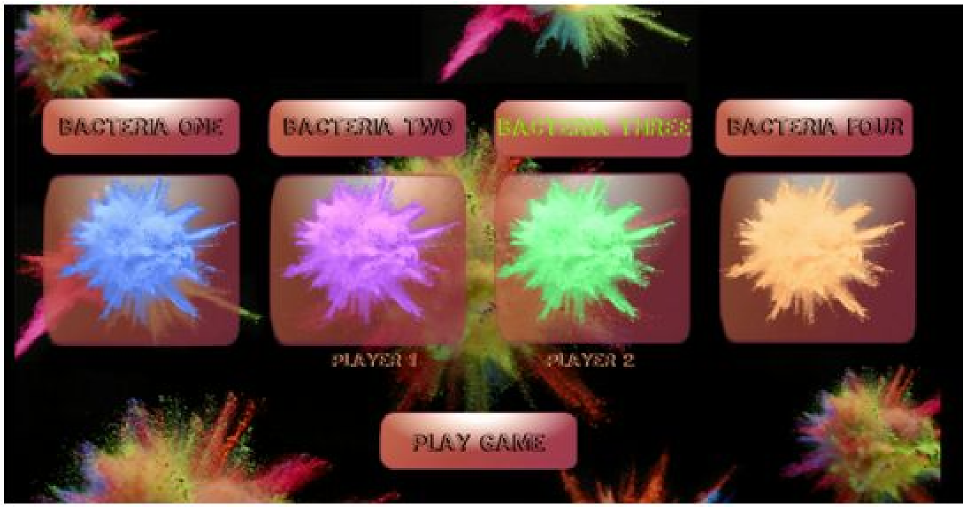
Main Menu screen

In this screen players can choose if they want to start the game and be assigned to random bacteria (*Play*), choose which bacteria they want to be and (*Choose Bacteria*) or *Exit* the game.



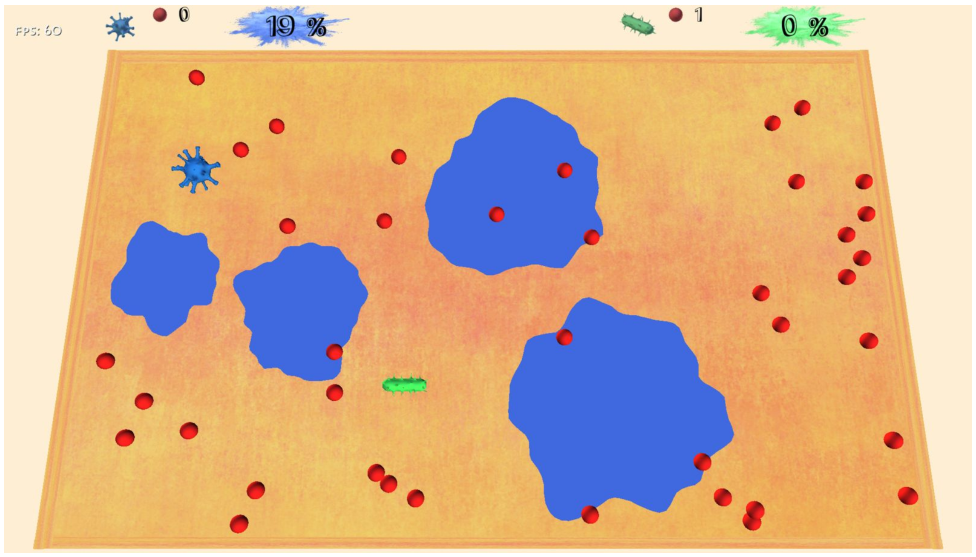
Choose Bacteria screen

In this screen, players can choose which bacteria they want to be. With their own gamepad, they can press the bacteria and then *Play Game* option, which bring them to the *Play* screen directly.



Play screen

This is the screen where the battle takes place. Players start in random position in the map and they all can move around, gather little pieces of the food they are conquering (in this case, as the map is a peach, little peaches) and explode!



Player movement

The player controls the target velocity vector of their character with the gamepad stick. We set a maximum velocity and an acceleration. Based the acceleration and the game time we move the current velocity in the direction of the target velocity (Figure 1). If the acceleration and the game time permit going over the target velocity, we clamp to it. This introduces a form of “inertia” that makes the movement more realistic.

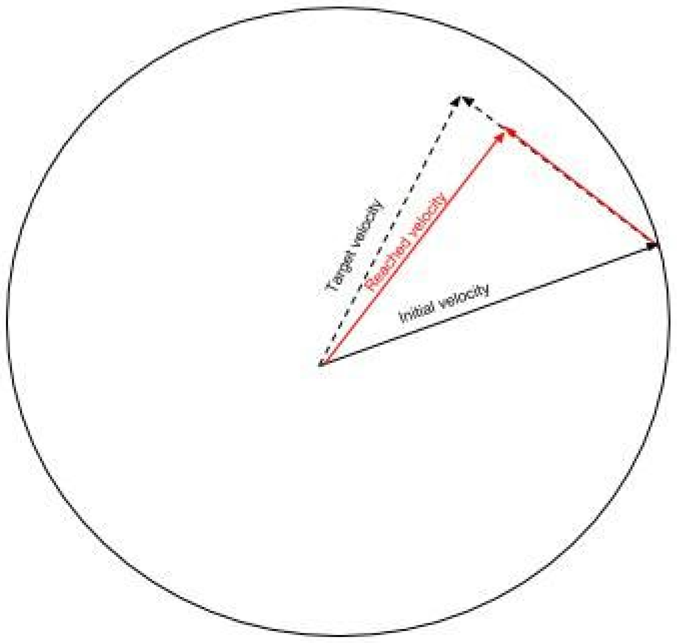


Figure 1: Velocity change

Collision detection and resolution

Collision Detection

In a typical situation, collision detection is achieved in two steps: a broad phase that aims to rule out impossible collisions in order to save computation time and a narrow phase that detects the collision at a detailed level. Given the fact that we only have at most four moving objects and they only have to be checked against themselves and a fairly small number of static objects, we decided to drop the broad phase and to only have a narrow phase that involves all the objects.

The movement of the players is limited to a 2D plane and 2D collisions proved to be satisfactory for our purposes. Because the meshes that we use don’t have large concavities, we base our approach solely on the projected convex hulls of those meshes. For this, we

first implemented Andrew’s monotone chain algorithm to quickly compute the convex hulls. In order to determine if two objects intersects, we check to see if their projected convex hulls overlap. This is achieved using the Gilbert-Johnson-Keerthi (GJK) algorithm.

The core idea of the algorithm is to leverage a property of the so called Minkowski Difference of the convex hulls (a Minkowski Sum between polygon A and polygon -B). This property states that two polygons overlap if and only if the Minkowski Difference contains the origin. Because the computational complexity of getting the actual difference is fairly large (O(n^2 log(n^2)), the algorithm employed checks for the origin containment by constructing simplices of up to 3 vertices (in 2D) based on support points on the Minkowski Difference in the direction of the origin. This is done until either the origin is contained in one such triangle(for the 2D method) or until it is proven the the origin can’t be reached.

GJK only tells us whether two convex polygons intersect, but in order to be able to resolve the collisions, we need to also find the smallest penetration distance and its direction. This is achieved using the Expanding Polytope Algorithm, which starts with the simplex provided by GJK. The penetration depth is the smallest distance from the origin to the Minkowski Difference. We find this out by constantly trying to expand the simplex with vertices on the difference that go away from the origin, until the closest simplex edge to the origin is an edge that belongs to the difference. The direction then is the normal to this edge, and the depth is the distance to the origin.

Collision Resolution

There are three types of visible collisions in the game:

1. Player vs player

   This is solved by moving the players away from each other, each by half of the penetration depth.

2. Player vs resource

   In this case, the resource vanishes.

3. Player vs walls

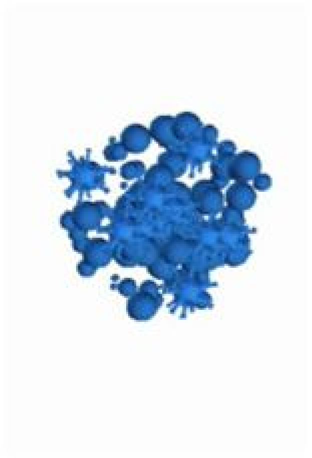
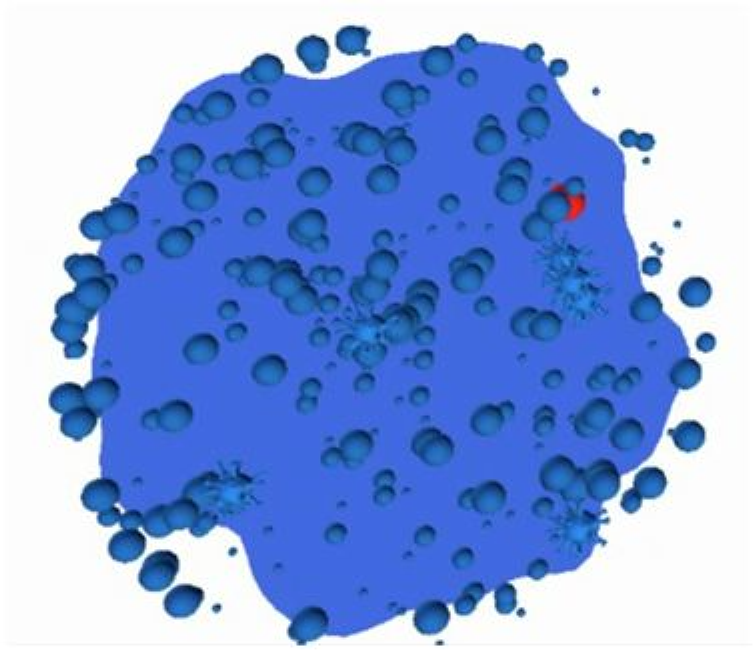
   Currently the velocity of the player is reflected against the penetration direction. This needs iterating upon, in order to achieve more “realistic” elastic collisions.

Explosions

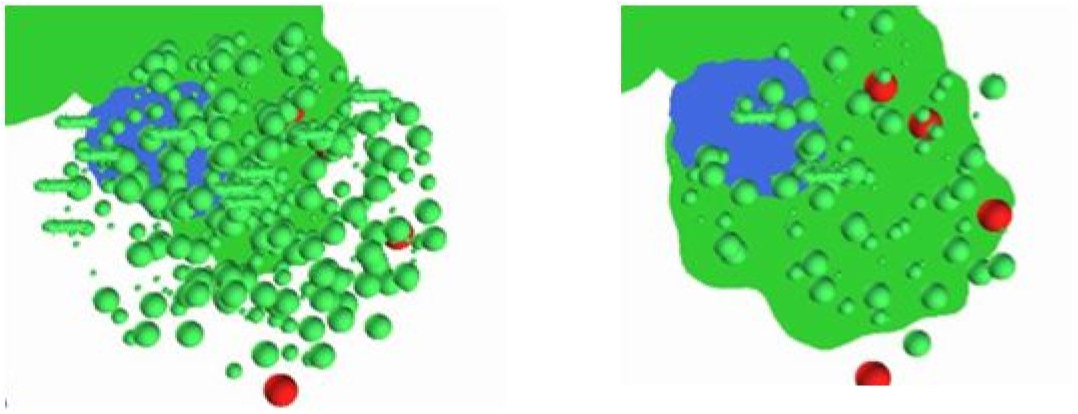
Particle System

To integrate explosion effects to the game, we tried to use several libraries that provide very nice 3D particles effects, as *Mercury Project* and *DPSF*. Monogame does not work with them, although they are made for XNA 4.0, and we couldn't find any way to fix it, so we decided to create our own Particle System in the game in order to generate explosions effects.

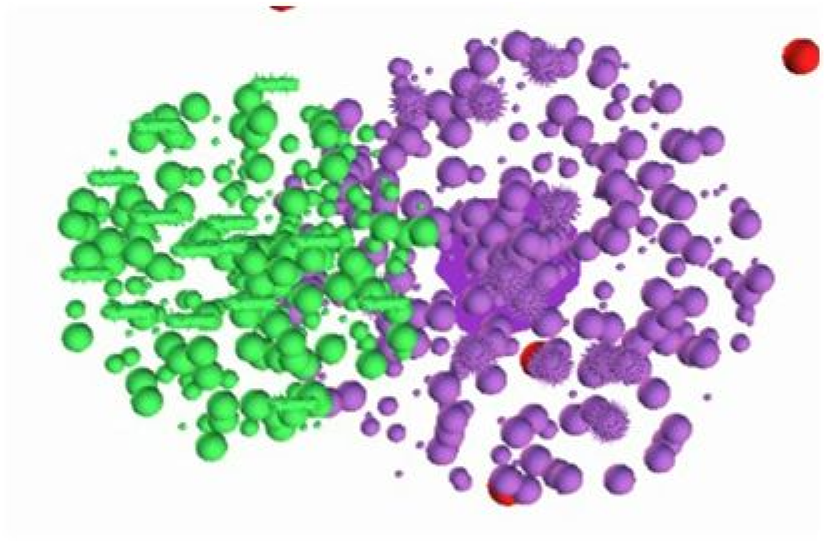
To integrate this explosions on the game, every time a player press the button to explode, the game takes the particle emitter and generates a new explosion with the corresponding particles (each player has different type of particles to generate different effects). In the figures below, there are some examples of little and a big explosion, and also how do they progress in time. We also use the bacteria mesh as a particle to simulate that is the bacteria itself that is exploding and staining the food below.

Little and big explosion of blue bacteria



Explosion at the beginning and the end of a green bacteria



Two bacteria exploding at the same time

Also, every time someone dies and spawns again in the screen, a little particle explosion appears to make it easy to the players to spot their bacteria in the map and continue playing

Bacteria appearing and moving with particles around

Floor stains

The stains are generated differently each time on the floor. The shape is generated using primitives and deformed consequently to look like a liquid stain.

The methodology for generating the shapes is called texture advection and proceeds as follows:

1. Generate a circle using triangle primitives placed as a triangle fan (currently using 100 triangles). (Fig. stainGeneration1))
2. Deform the outer vertices of the fan inside and out. The deformation is done in the direction of the vector (*outerVertex* - *center*).
   How much should these vertices be displaced? This amount is given by a generated 2D texture that contains noise information (see Fig.stainGeneration2). Each vertex of each triangle is deformed according to each value on a random row of the texture. The texture is 100px x 100px and there are 100 vertices on the triangle fan. There is a one-to-one correspondence and thanks to the tillable property of the texture, the value on the left of the row are related with the values on the right. This makes that the first vertex and the last one are connected smoothly.

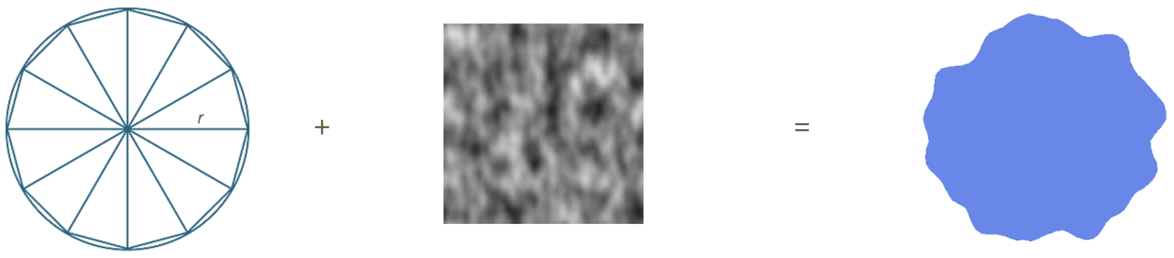
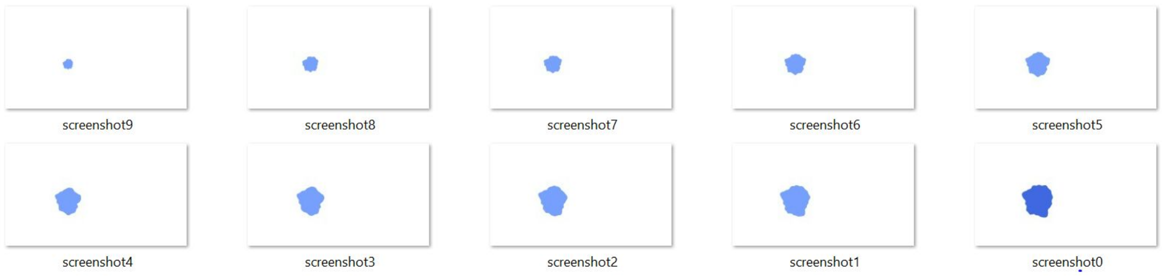


Fig. stainGeneration1 Fig. stainGeneration2

As a random row of the 2D texture is used to deform the stain each time, a new shape will be generated for each explosion. Then, we'll have as many different stains as we want! We'll have as many shapes as number of rows on the noise texture.

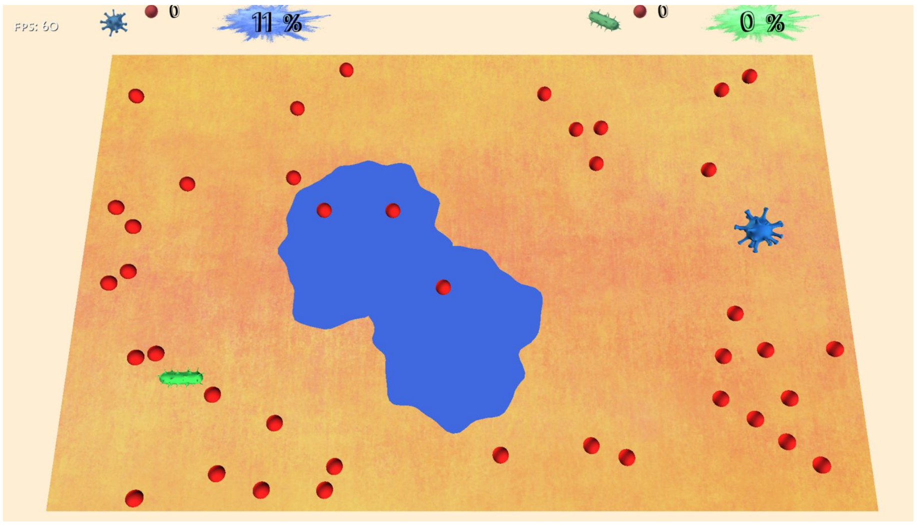
Stains do not appear still, they grow gradually! When a stain is generated, different sizes are created until it reaches the actual radius size. This makes it to grow gradually.



Scoring

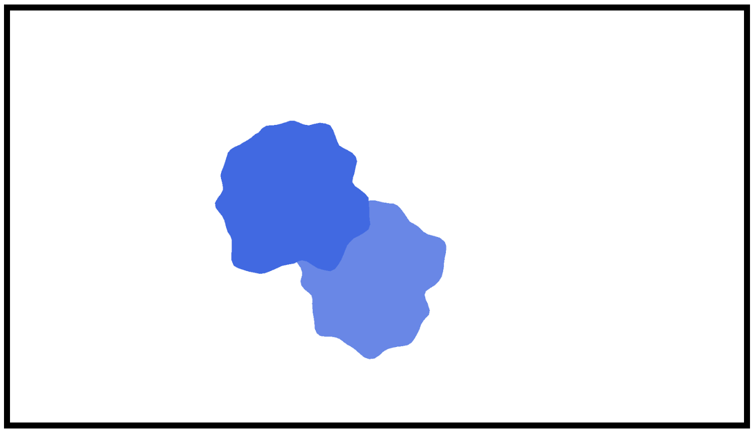
Scoring is not shaped-based anymore. Previously we used an approach based on the geometry of the stains. As they were basic 2D circles, it was easy to compute the area and add/ subtract the overlapping areas.

As the stains are not round anymore, the approach is completely different. The principal idea is to count the area stained by each player. Each time a new stain is generated, the proportional area that corresponds to this stain is added to the score of the player. Overlapping has to be considered! Imagine that a player did already stain the floor on an area. It explodes again next to it, and only the region of the new stain should be added to the score (see image below).



It is difficult to say like this, which is the region corresponding to the new stain.

How to do this? The stains on the board are rendered from top-down view to a texture. This view enables to keep track of the score of the players. The score is computed by counting pixels on this view. Checking the whole board every time is really expensive, so instead: for every new explosion, only the new stain is checked.



How to differentiate the new stain from the old ones? The new has a color property that makes it different from the old stains: it's Color.A is lower than the rest (Colors have 4 channels: RGBA). After having added the score, the Color.A property of the new stain is changed to the same value as the other stains. With this, the new stain becomes part of the old stains present on the board.

Player Interactions

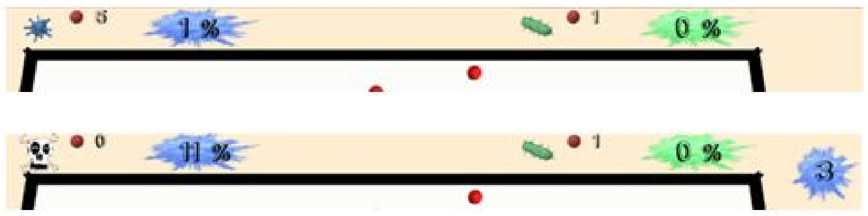
The goal of each player is to conquer the maximum possible part of the map by exploding and colouring the floor with its bacteria colour. To do that, each player can gather resources, and the more resources you have, the bigger will be the explosion. However, other players can kill you and make you loose all the resources you had gathered! If another player explodes and you happen to be in its explosion area, you are dead! That doesn't mean you can't continue playing, but you will have to wait a spawn time and then your bacteria will randomly appear again in the map. The more resources you gathered, the bigger is the radius of your explosion and indeed, the waiting time to be respawned will be bigger :(

Also, players must be careful to not be killed too many times, because every three kills you will have an extra seconds punishment and you will spawn later (detailed explanation in Score panel part).

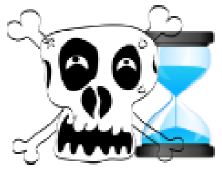
When you explode or get killed, all the resources you gathered are respawned gradually all over the map. Not in the previous position, but in a new one!

Score panel

Players can always look at their scores on the top of the screen. Each player has its bacteria at the left of their mini panel, indicating their panel position. When a player is dead, the bacteria image is replaced by a skull during the time it takes the player to spawn again in the map.



Also, while a player is dead, a counter of the remaining seconds it will have to wait until appearing again is shown at the right part of the screen. If other players are also dead, all the counters will appear (in the same order as they are at the top) and players will be able to see who is going to be alive first and prepare theirs strategies!



If a players gets killed (by other players explosions or, in future version, dashed) too many times, it will receive a punishment. This is shown on the second row if the score panel. Every three kills, a skull with an hourglass will appear. This punishment works as follows:

- 0 Skulls: Player has been killed less than 3 times. Time to spawn: 2 s.
- 1 Skull: Player has been killed more than 3 times. Time to spawn: 4 s.
- 1 Skulls: Player has been killed more than 6 times. Time to spawn: 6 s.
- 3 Skulls: Player has been killed more than 9 times. Time to spawn: 8 s.

Finally, the total score of each players appear in the explosion of their colour, at the top and right part of their own panel. It is counted in % of the map that each player has coloured, and will change everytime you explode with respect of your conquered percentage.