Team #3
Evan Gofourth
Katharine Wu
Jacob Pfeiffer
Elena Zavala
Alicia Zavala

Project Name:
Code Clash

Project Synopsis:
A game for kids to start learning basic programming logic in a player versus player game, where they can program their own bots (spaceships!) to mine resources (asteroids).

Project Description:
There exist a plethora of educational programming tools for kids with an interest in programming to learn. However, there lacks a concrete goal to the educational tasks. Learning for the sake of learning fails to captivate the attention of children for long enough to make *real* progress. Code Clash seeks to resolve this issue by providing new programmers with a Visual Programming interface that makes writing a program *easy*. Once a user is done writing a script, they attach it to a unit in a 2D grid based game and see how that unit behaves. The player's units (that behave according to scripts written by the player) will compete head to head against other players and their scripted units in an online arena. This way competition motivates the player to learn and write better scripts, rather than a dull prompt that demands the user complete a programming task.

Project Milestones:

Fall:

October 26: Project Proposal
October 26: Project Proposal Video
November 20: Have a multiplayer working proof of concept (something that works, if clunky)

Spring:

Feb 28: Have a polished system for finding and creating online games.
April 15: Have multiple bot commands made and implementable by the player(s)
April 15: Have a streamlined easy to use VPL editor.
April 30: Replace all filler art with finalized, professional looking artwork.
May ?: Deliver final working product and finish final project video

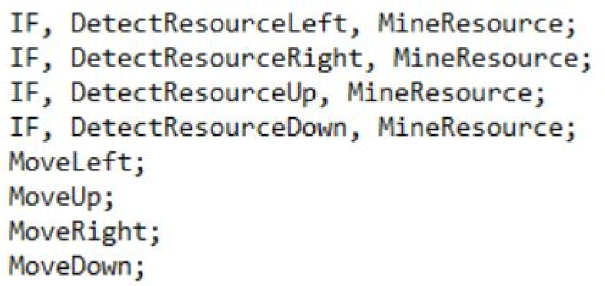
Budget:

Item: Unity API/Software	Projected cost: $0
Item: Mirror Software (Unet copy)	Projected cost: $0
Item: Professional Artwork	Maximum Budget: $50

Preliminary Project Design:

A typical use case -

In the typical use case, a user will launch the game and appear at the main menu. They will enter the ‘code editor’ and write a behavioural script for a robot agent using visual coding logic blocks (see Figure 2). When they finish, the GameObjects (logic blocks) the player has positioned in a Unity scene (the editor) will be translated into text based code, which will adhere strictly to a format we have specified.



```
IF, DetectResourceLeft, MineResource;
IF, DetectResourceRight, MineResource;
IF, DetectResourceUp, MineResource;
IF, DetectResourceDown, MineResource;
MoveLeft;
MoveUp;
MoveRight;
MoveDown;
```

Figure 1 (Code produced by our VPL editor)

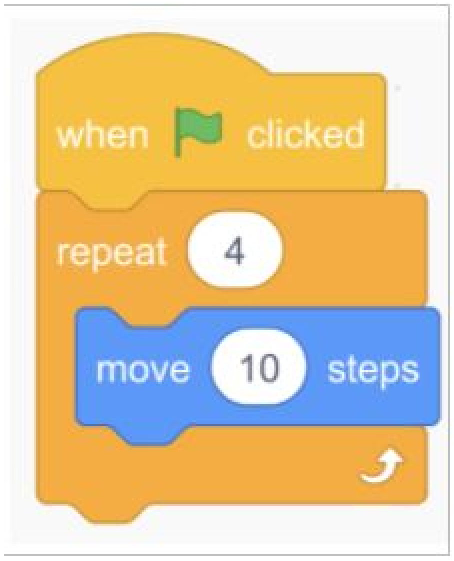
Once the code has been generated, the user will join or host a multiplayer game. Before entering the game, they will be asked to attach behaviour scripts they have written to each of their unit types. For example, the script shown in Figure 1 may be saved under the name “MiningScript”, and then selected from a number of scripts in a drop down beneath the “MiningDroid” unit type to act as that unit type’s behaviour script for the game.

When the game begins, each player will have only one control ship, which is capable of spawning new units. Spawning a new unit will cost resources, and consume an action point from the control ship (this limits actions taken by a unit per turn). The control ship will spawn units as the player has programmed the control ship type to do. Each spawned ship will behave as the player has programmed that ship type to behave.

Ideally, these spawned units will gather resources, attack enemy ships, and protect the control ship for 1000 turns. At the end of 1000 turns, the player with the most resources wins.

How our editor works -

Users write code by dragging and positioning visual logic blocks. It will, in general structure, resemble the drag and drop visual coding system used by MIT’s Scratch Visual Programming Language.



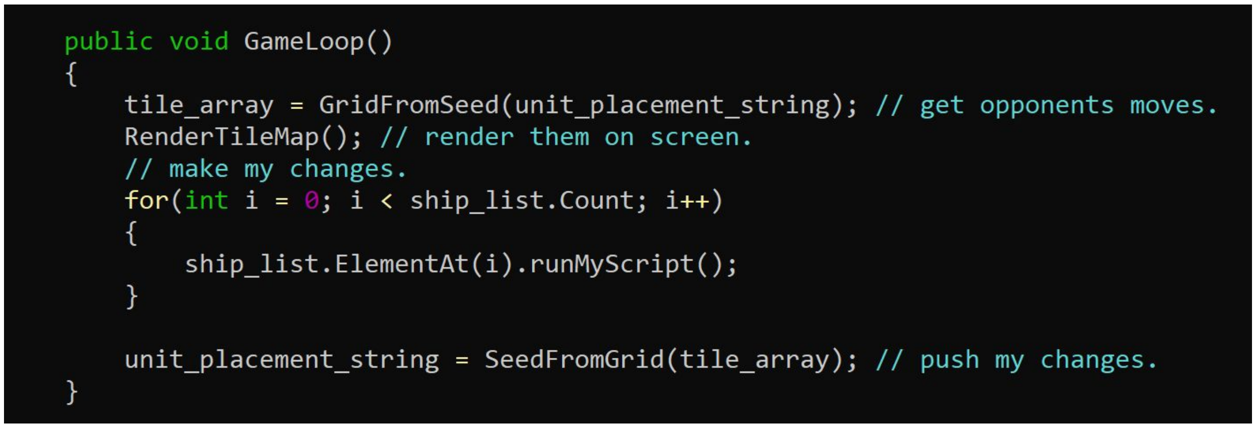
https://www.google.com/search?q=mit+scratch+code+block&rlz=1C1CHBF_enUS841US841&source=lnms&tbm=isch&sa=X&ved=2ahUKEwi58NqW_tDsAhVBKKwKHYKNB7cQ_AUoAXoECBoQAw&biw=1536&bih=730#imgrc=eYsMAsP-Smz9GM
Figure 2 (Scratch Visual Logic Blocks)

The produced code will be composed of conditional statements, loop statements, and C# method calls. For example, in figure 1, DetectResourceLeft is a precompiled, private C# method. It is a member of the 'Ship' class, and will be called by the ship this script is attached to once each turn. If the method returns 'True', then the C# method 'MineResource' will be called. If MineResource is called, the player will gain some number of resource points, and that ship's script will end, as it has consumed its action point (default is one per turn).

By allowing each ship to make C# method calls based on those generated by a user written script, we effectively allow our players to program the behaviour of agents in a 2D grid world, which was the ultimate goal.

Networking with Mirror - https://mirror-networking.com/

To allow competition, which we feel is imperative to learning, the players units will be competing against another player's units. Networking is handled by Mirror, the successor to Unity's Unet networking technology. We will be using Mirror's 'Telepathy Transport' option for brokering games between clients. Mirror's website describes Telepathy as "Simple, message based, MMO Scale TCP networking in C#". We will synchronize game states using our own 2D grid 'unit position' seed system. We produce a list of characters representing tile types, unit types, and unit color (red or blue depending on which player they belong to). We maintain order in this list, convert it to a semicolon delimited string, and synchronize this string for all players after each turn using Mirror's SyncVar. Each turn a player will load their grid from this synchronized string, run each of their unit's scripts (which alter the grid), then push their changes to the synchronized string. This is the primary game loop.



```
public void GameLoop()
{
    tile_array = GridFromSeed(unit_placement_string); // get opponents moves.
    RenderTileMap(); // render them on screen.
    // make my changes.
    for(int i = 0; i < ship_list.Count; i++)
    {
        ship_list.ElementAt(i).runMyScript();
    }

    unit_placement_string = SeedFromGrid(tile_array); // push my changes.
}
```

Figure 3 (Primary Game Loop)

Technical Constraints -

Because we are using a compiled programming language (C#) and want our product to be available as a compiled executable, we are unable to utilize the power of an interpreter to 'evaluate' parsed and sanitized user generated code. This means all the logic blocks must have implementations at compile time rather than at run time. We have decided this is a tolerable constraint, and also forces us to have a safer implementation (no user written code execution).

Ethical Issues:

There is an active debate as to whether or not 'violent video games' are ethical. Some have even proposed that such games can be harmful to the mental health of children. Our product is targeted toward children and intended to be educational. This being the case, we should do everything in our power to strip away any human elements of violence in our game. Tomohiro Nishikado, creator of the popular game Space Invaders, shared these concerns. It was for this reason that he made sure all violence in his game was directed toward 'aliens'. This need for our game to be void of human violence is part of the reason for our choosing 'robot' agents for players to program. This way when robot agents 'destroy' one another, there is no suggestion of human violence.

MIT's Battlecode, an inspiration for our game, did (and does) include elements of human violence. While this may be acceptable in their product, which is marketed toward college age users, we want to keep these themes out of our game which has a much younger target demographic.

Intellectual Property Issues:

Our app is inspired by **MIT’s Battlecode** game - where players program AI units to compete in a 2D grid world. They use their game in annual coding competitions. It is perfectly legal for us to ‘clone’ this game (or build a game that resembles it) so long as we don’t steal artwork, code, or backend services. Battlecode’s theme changes every year, so our static space theme should not be an issue of intellectual property. Much of our artwork we plan to make ourselves, the rest we will get from independent artists and use according to the license. Most of the artwork we have sourced is licensed under the Creative Commons License. To our knowledge, we are using very different backend technologies from those employed by MIT. It is our understanding that Battlecode exists as a webapp (built with JavaScript), whereas Code Clash will be a Unity C# precompiled application. There may later exist a WebGL build.

Our Visual Programming Language is inspired by **MIT’s Scratch**. Scratch is a visual programming language that utilizes drag and drop ‘logic blocks’. Lots of other VPL’s utilize this drag and drop code block system. While the method of writing visual code will be the same, our logic blocks, their purposes, and their style will differ sufficiently from MIT’s Scratch. Our final product should differ more from MIT’s Scratch than does Snap! (another popular VPL).

Changelog:

The project synopsis was updated to no longer refer to the game as being ‘online’. While there will be multiplayer matchmaking, the plan is no longer for the game to exist as a webapp, but rather as a compiled executable. Similar changes were made in the project description.

The November 20th milestone was updated to state that we expect to have a multiplayer working proof of concept rather than a single player one. This is because implementing multiplayer went smoother than expected.

The February 28th milestone was updated. It formerly was to implement two player networking, but this is largely already accomplished. We will instead aim to have a very polished system for finding and creating multiplayer games rather than just introducing the feature.

Two more milestones were added to the Spring semester: one on April 15th and one on April 30th. These are dates by which we plan to have a streamlined VPL editor and professional artwork respectively.

Work Plan:

Evan Gofourth	Unity/Mirror lead engineer
Jacob Pfeffer	Frontend and backend engineer
Katharine Wu	Frontend and backend engineer
Alicia Zavala	Frontend engineer
Elena Zavala	Frontend engineer

		Week of October 25th	Week of November 1st	Week of November 8th	Week of November 15th	Week of November 22nd	Week of November 29th	Week of January 31st	Week of February 7th	Week of February 14th	Week of February 21st	Week of February 28th	Week of March 7th	Week of March 14th	Week of March 21st	Week of March 28th	Week of April 4th	Week of April 11th	Week of April 18th	Week of April 25th	Week of May 2nd
	Dev	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa	Su M Tu W Th F Sa
Have a multiplayer working proof of concept (something that works, if clunky) (80 hrs)	Evan	4	4	4	4																
	Katharine	4	4	4	4																
	Jacob	4	4	4	4																
	Ali	4	4	4	4																
	Elena	4	4	4	4																
Have a polished system for finding and creating online games. (120hrs)	Evan					4	4	4	4	4	4										
	Katharine					4	4	4	4	4	4										
	Jacob					4	4	4	4	4	4										
	Ali					4	4	4	4	4	4										
	Elena					4	4	4	4	4	4										
Have multiple bot commands made and implementable by the player(s) (32 hrs)	Ali											4	4	4	4						
	Elena											4	4	4	4						
Have a streamlined easy to use VPL editor. (108 hrs)	Evan											4	4	4	4	4	4	4			
	Katharine											4	4	4	4	4	4	4			
	Jacob											4	4	4	4	4	4	4			
	Ali															4	4	4			
	Elena															4	4	4			
Replace all filler art with finalized, professional looking artwork. (16 hrs)	Ali																		4	4	
	Elena																		4	4	
Deliver final working product and finish final project video (44 hrs)	Evan																		4	4	4
	Katharine																		4	4	4
	Jacob																		4	4	4
	Ali																				4
	Elena																				4