# SquareBattle

## At a Glance

SquareBattle is a game written in LabVIEW where you program your own Squares to compete against other peoples Squares in the arena. You win by having the most Squares at the end of the game.

## Goals

1. For everyone to have fun!
   The idea is that everybody gets a chance to do some coding and strategizing if they want, and if they don't want it will still be fun and educational to watch.

2. Simple and Accessible
   It is simple enough that a novice can set up a Player that does basic actions and open-ended enough that experts can utilize all of their LabVIEW knowledge to make things too complex and end up getting beaten by the novices. It also provides an introduction to object oriented programming without the need to understand it fully.

## TLDR

In the SquareBattle project you need to build your class in the Squares folder and make sure that it inherits from “Square.lvclass”. Override “Player Info.vi” in any classes you want to show up in the list of Starting Players. “Run.vi” is called once for each Square you have on the board. The Arena, or map is a 2D array where each location is marked with a -1 (empty location) or the team number of the square occupying that location. You can “see” the 8 squares immediately adjacent to your player as an input to Run.vi. The goal is to take that information and decide whether to “Wait” which doesn’t do anything this Round, “Move” in the direction you supply, “Attack” in the direction you supply, or “Replicate” in the direction and using the class that you supply. In other words, when replicating, you can specify any descendent of the Square class to replicate into, so you may start out with a class that replicates a lot but then sends out attackers to do battle.

There are three pieces of global data that you can get by dropping the vi from the Square.lvclass into your vi.

1. Round.vi – The current Round of the battle.
2. Board Size.vi – The number of columns and the number of rows (only one number because the Arena is a square)
3. Stats.vi – Arrays containing statistics for the Teams currently competing. These arrays can be indexed using the team number.

## The Nuts and Bolts

Here are some terms that we will use as they apply to SquareBattle:

- **Coder** - Anyone who creates a Player to compete in SquareBattle. There are actually many, many things you can call these people (enthusiasts, software programmers, gamers, wicked awesome etc.) but I will try to use the term Coder for the purposes of this document. A Coder might look something like this:

- **Arena** - The area of the board where the Players compete against each other. The Arena is really just a 2D array of numbers. An empty Arena of size 5 (height and width) would look like this:

| | | | | |
|---|---|---|---|---|
| **-1** | **-1** | **-1** | **-1** | **-1** |
| **-1** | **-1** | **-1** | **-1** | **-1** |
| **-1** | **-1** | **-1** | **-1** | **-1** |
| **-1** | **-1** | **-1** | **-1** | **-1** |
| **-1** | **-1** | **-1** | **-1** | **-1** |

The Arena wraps around the edges, so if a Square is in the final column and moves off the board to the right (East) it will appear on the board in the same row but in the first column on the left. Each location in the Arena has a position made up of a Row and Column with Row 0 along the bottom and Column 0 along the left of the Arena. Each square is given its row and column each time it is called.

The same 5x5 example with the column and row indicated for each location

| | | | | |
|---|---|---|---|---|
| **0,4** | **1,4** | **2,4** | **3,4** | **4,4** |
| **0,3** | **1,3** | **2,3** | **3,3** | **4,3** |
| **0,2** | **1,2** | **2,2** | **3,2** | **4,2** |
| **0,1** | **1,1** | **2,1** | **3,1** | **4,1** |
| **0,0** | **1,0** | **2,0** | **3,0** | **4,0** |

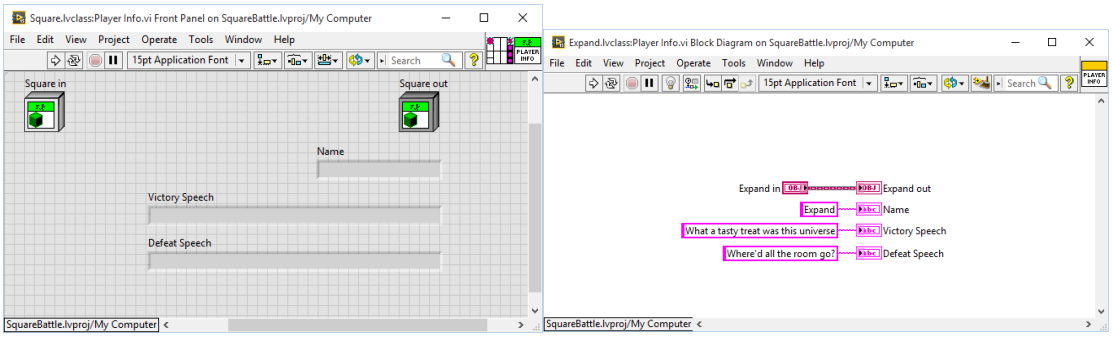
- **Game Engine** – The Game Engine was developed by Endigit and takes care of updating the Arena map and processing “Rounds” (more details below).
- **Player** - A child class of "Square.lvclass", developed by a Coder which will either start out in the arena or will be spawned into the arena by a different Player on the same Team. A Player can do one of 4 things each “Round”:

"Wait", "Move", "Attack" or "Replicate"

- **Starting Player** - A Player that has been assigned a name in "Player Info.vi" by its Coder will show up in the list of available starting players and will be placed randomly in the Arena at the start of the Battle. An Arena with a couple of starting players might look something like this:

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

- **Team** - The starting Player and any Squares spawned from it, all of which share the same color.
- **Square** - An individual instance of a Player that occupies one and only one location in the Arena. Because a Square can replicate, a Team is made up of more than one Square. Because a Square can replicate to produce a Square that is a different Player (ideally made by the same Coder), a Team can be made up of several Players. For instance, a Starting Player, called "Tweedle-Dee" might have code that checks to see if there are enemy Squares nearby and then, if there aren't any, replicate in a given direction. But, when "Tweedle-Dee" replicates it spawns a "Tweedle-Dum" that has completely different code and that doesn't even check for enemies, but just moves around in circles.
- **Round** – A Round is one round of the Battle. Every Square's "Run.vi" is called, the results are processed by the Game Engine, and the map is updated by the Game Engine. When we talk about the Coder creating a Player, we are really talking about them creating a new Class that inherits from "Square.lvclass" and overwriting "Run.vi". We will talk more about how to do that below, but suffice it to say that each Square has its own "Run.vi" that is called every Round of the game as long as the Square is still alive.
- **Battle** - The battle starts by putting all of the selected Teams randomly on the board and when the Battle ends, the Team with the most Squares on the board wins and gives its Victory Speech.
- **Victory Speech** - A text string assigned in the "Player Info.vi" of a Starting Player that appears when that Team wins the Battle.
- **Defeat Speech** – A text string assigned in the "Player Info.vi" of a Starting Player that appears where the team name is in the list of teams when that team is defeated.
- **A Square's Code** – There are three vi's, two of which have already been mentioned, that a Coder needs to be aware of. All three are vi's for override from the parent Square.lvclass. Once a Player is set to inherit from Square.lvclass, these three vi's will appear under the "New->Vi for Override" menu.




**Player Info.vi**

Player Info.vi simply has three outputs, “Name”, “Victory Speech” and “Defeat Speech” that the Coder can wire strings up to and that, when provided, allow the Player to show up in the drop-down list of starting players and provide the text for the victory speech and defeat speech.

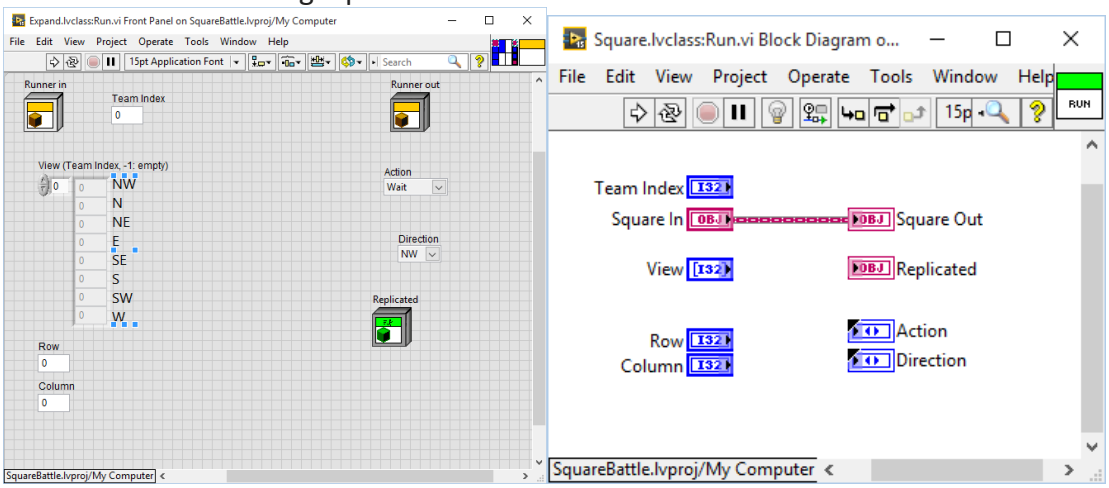
**Eliminated.vi**

This vi is run once for each square that is defeated and has the following inputs:

1. Team – This is the Square’s own team number.
2. Killer – This is the name of the team that the square that killed you came from. You should go after this team… they might be dangerous!
   **Note:** If there were more than one teams attacking the location the square died in or if there were more than two teams in the same location at the end of the round, the team that gets the kill is essentially random.
3. Row – The Square’s row when he died.
4. Column – The Square’s column when he died.

**Run.vi**

Run.vi has the following inputs:




1. Runner in – This object contains the cluster of class private data and can be used to store anything that you want to have access to from one round to the next.
2. Team Index – This is the number assigned to your team before the battle starts.

3. View (A chunk of the Map) – This is a 1x8 1D showing what the arena looks like in the locations adjacent to the square starting in the NW corner and going clockwise. Other Squares that are on the same Team show up as the same number as your Team Index. Other Squares that are on different Teams (sometimes referred to as enemies) show up as their own Team Indexes. The array can be indexed using the "Direction" enum to see what is in a given direction.
   Given the following Arena:

| | | | | |
|---|---|---|---|---|
| -1 | -1 | 0 | -1 | -1 |
| -1 | 0 | -1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

   Here is the chunk of the Map that the red **0** Square in the 1th column would be given as an input to "Run.vi":

| |
|---|
| -1 |
| -1 |
| 0 |
| -1 |
| 1 |
| -1 |
| -1 |
| -1 |

   And here is the chunk of the Map that the blue Square would be given:

| |
|---|
| 0 |
| -1 |
| -1 |
| -1 |
| -1 |
| -1 |
| -1 |
| -1 |

4. Row – The Square's current row in the Arena. This would be "3" for the red **0** Square in the 1th column in the above example.
5. Column – The Square's current column in the Arena. This would be "1" for the red **0** Square in the 1th column in the above example.
6. Remember that in addition to those inputs you also have access to the current round number and the board size by dropping "Round.vi" or "Board Size.vi" into your "Run.vi". Both are found in the "Square.lvclass" class in the project library, or by typing "Round.vi" or "Board Size.vi" into quickdrop (;

Run.vi has the following outputs:

1. Square Out – Passes out the class private data of the Player class. Can be used to pass data from one round to the next. Because this output is not Dynamic Output, the Player can be changed from one round to the next.
2. Action – Enum of “Wait”, “Move”, “Attack”, or “Replicate”. Wait, Move and Attack each take 1 Round. Replicate takes 100 Rounds.
3. Direction – Enum of “NW”, “N”, “NE”, “E”, “SE”, “S”, “SW”, or “W”. Because this is the same order as the map is received, a coder can index the View using Direction to see what is in that direction.
4. Replicated – Class of type “Square.” When replicating, this will determine the type and the starting data of the new Square.

**Waiting**

If a Square outputs “Wait” for its Action, the other outputs, “Direction” and “Replicated”, are ignored.

If this is what the board looks like at the start of the round:

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

And both Squares output “Wait” as their action… This is what the board would look like after waiting…

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

which is kind of boring…

**Moving**

If a Square outputs “Move” for its Action, the “Direction” output is used but the “Replicated” output is still ignored.

If this is what the board looks like at the start of the round:

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

And both Squares output “Move” as their action and “NE” as their direction… This is what the board would look like after moving…

| | | | | |
|---|---|---|---|---|
| -1 | -1 | 0 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

And if they moved in that direction again…

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 0 | -1 |

**Attacking**

If a Square outputs “Attack” for its Action, the “Direction” output is used but the “Replicated” output is still ignored.

If this is what the board looks like at the start of the round:

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

And both Squares output “Attack” as their action and “NE” as their direction… This is what the board would look like after Attacking…

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

which is kind of boring…

OK, here is the scoop. Attacking is just like waiting, unless there is another Square in the “Attacked” location at the end of the Round. You may make it part of your strategy to always use “Attack” instead of “Wait” because then you can preemptively take out any square that moves into that space. Unfortunately, if another Square from your own Team moves into that space, they will no longer exist.

Here is a better example…

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

0 = “Move” “SE”
1 = “Attack” “NW”

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

No more TweedleDum. The Game Engine gets everybody’s moves and then processes them at the end of the Round. At the end of the Round, TweedleDum was in a square that was being attacked, and anybody that is in a square that is being attacked at the end of a Round, dies.

A potentially more common scenario would be that two enemies encounter each other. In this situation they may both have code that essentially says “If you see an enemy, attack in that direction” in this case both of the Squares would be in a square that is being attacked at the end of the Round and both would die.

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | -1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

0 – “Attack” “SE”
1 – “Attack” “NW”

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

It’s a tie!! I think the game engine would pick a random winner (:

**Dying**

Before leaving this morbid topic of dying Squares, now might be a good time to mention that there are two ways for you Squares to perish:

1. Being in an "attacked" location at the end of a Round.
2. Being in the same location as another Square at the end of a Round.

Because of way to die #2, it is a good idea to try not to have two of the same Team's squares move into the same location, which is much more difficult than it sounds since unless the Coder programs in some way for the squares to share each other's positions, each square is only aware of its immediate surroundings. Keep your eye on the "Collision" statistic to see how much you are killing yourself off.

**Replicating**

If a "TweedleDum" Square outputs a "TweedleDee" class to the Replicated output and a value of "Replicate" to the Action output and a value of "E" to the Direction output, a "TweedleDee" Square would spawn in the location adjacent to "TweedleDum" in the East direction. Replicating takes 100 Rounds, so TweedleDee would appear in the direction indicated and then both TweedleDum and TweedleDee would sit there vulnerable for 100 Rounds. Then their "Run.vi"s would be called again and the play would continue.

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 |

**0** – "Replicate", "E", "**0**" (another Class inherited from "Square.lvclass" created by the same Coder)
1 – "Replicate", "E", "1" (another instance of the same class)

| | | | | |
|---|---|---|---|---|
| -1 | -1 | -1 | -1 | -1 |
| -1 | 0 | 0 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | 1 |
| -1 | -1 | -1 | -1 | -1 |

**0** = TweedleDum
**0** = TweedleDee

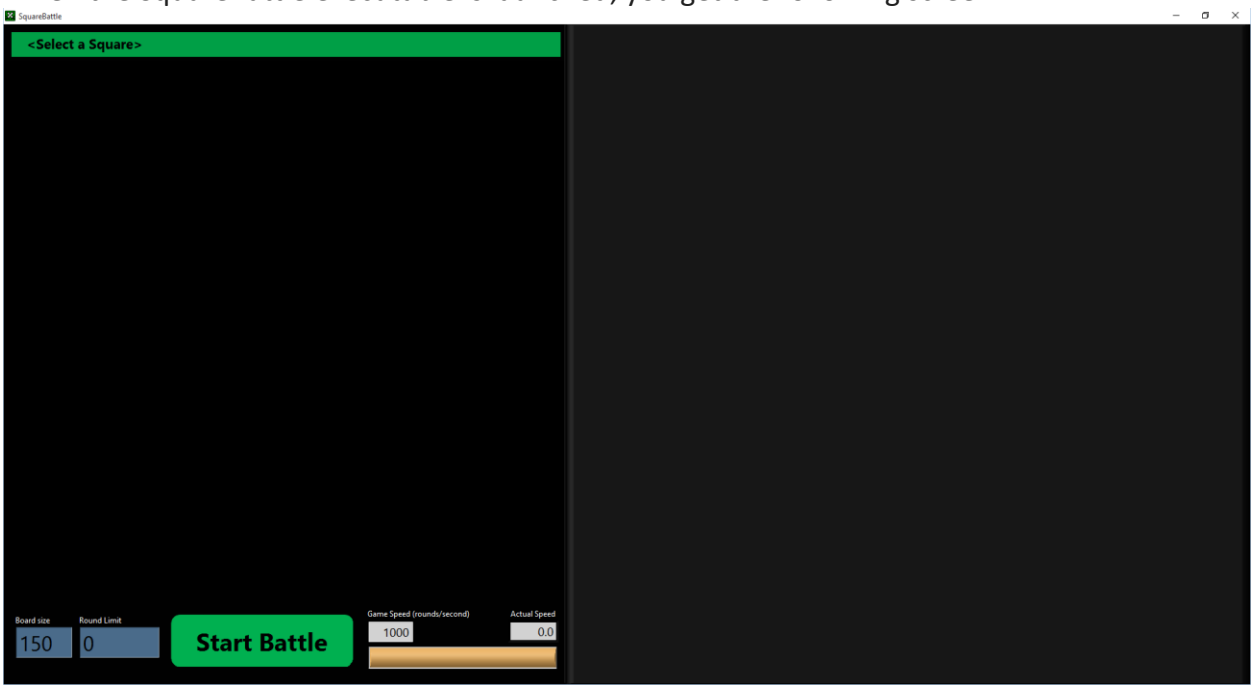
**Reminder:** There are three additional pieces of data that are meant to be used by the Coders when creating Teams of Players:

1. Board Size – The number of Columns and the number of Rows in the square Arena (It's a square, so they are the same!)
2. Round – The current Round of the Game. The game engine increments this by one every round of the game.
3. Stats – Info about each of the teams. You might use these stats to change your strategy based on how fast your opponent is replicating or how close you are to wiping out your opponent.

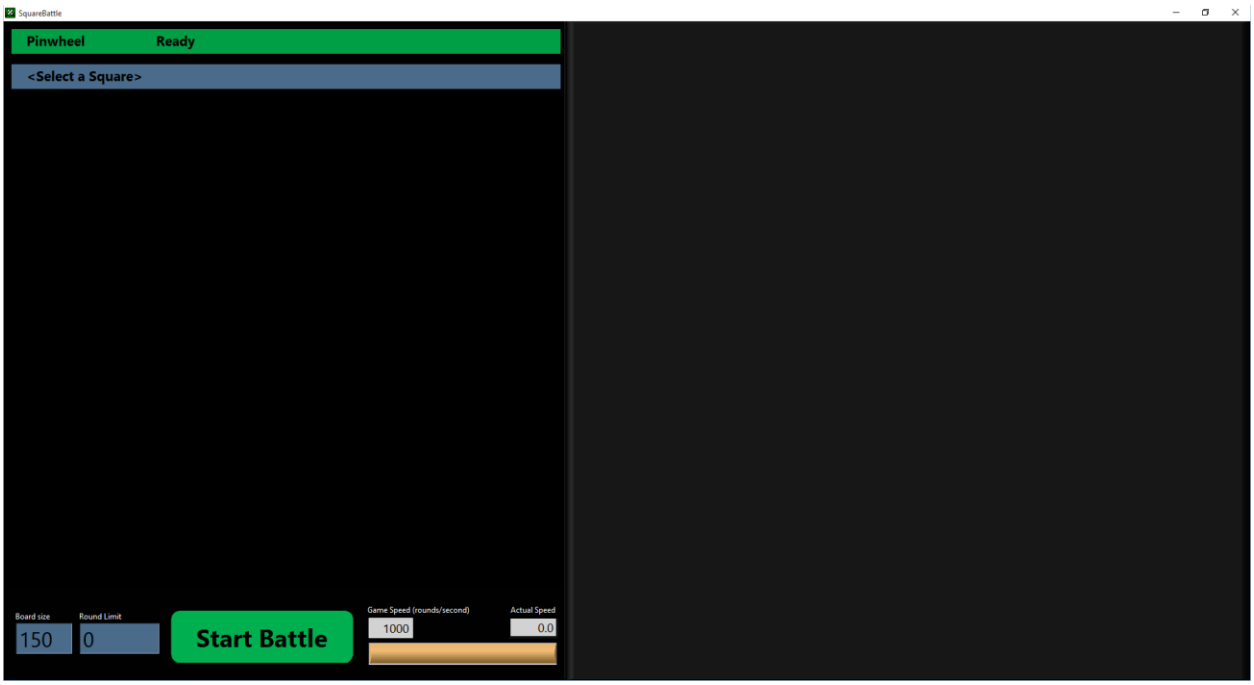
These vi's are found in the "Square.lvclass" and you can drop them into your code wherever you want.

## Running SquareBattle

When the SquareBattle executable is launched, you get the following screen:



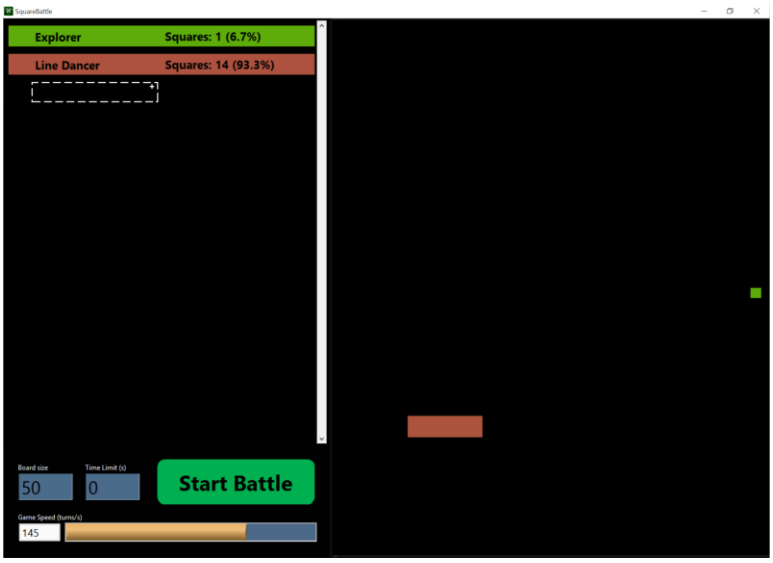

**Player Selection** - This is the area in the top left of the screen where it states "<Select a Square>". Click to select the starting squares. The list of available squares is auto-populated by searching through a folder on disk called "Squares" (that resides in the same directory as the executable) and looking for all of the squares that people have created. The players you select will be given a random color which will be the color of the squares during gameplay. In the following image, I have selected the "Pinwheel" as the first player and the game automatically assigned Pinwheel the color green (the color can be changed by clicking to the left of the player name).



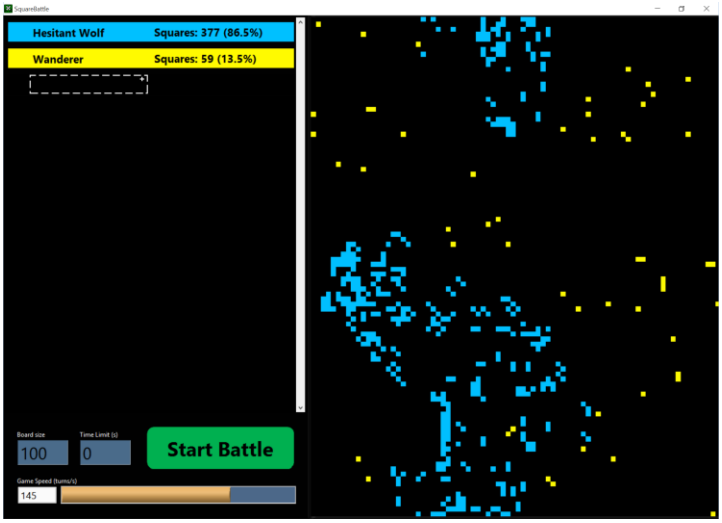

**Board Size** - Specifies how many rows and columns there are in the Arena. The Arena is square, so setting Board Size to 150 would give a 150x150 sized board.
**Time Limit (s)** - If you set this to anything other than 0, it counts down and the game ends when it gets to zero.
**Start Battle button** - After you have selected your players you hit this button to start the Battle.
**Game Speed (turns/s)** - This throttles how fast the game is played. If you throttle the game down to 1 turn/s, it will wait 1 second before the next Round.
**Actual Speed** – Because of the time it takes to run through every Square's code, the actual speed might be less than what the Game Speed is set to.
**The Arena** - This is the area that takes up the rest of the screen on the right. This is where the squares will be shown. It is an intensity graph where each "Pixel" in the graph is a square. The squares are differentiated by Team based on their color. There is an "Update FP" part of the SquareBattle code that updates the Arena based on the 2D array Arena map for your viewing pleasure.

## SquareBattle - A Couple of Examples

Note: These examples are from an earlier version of the game. While we have migrated some of the squares over to the newer revision, they might not all be available currently.

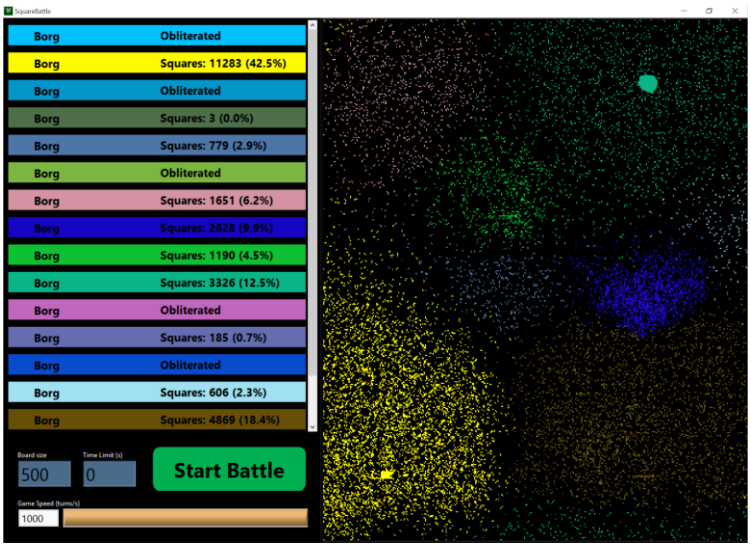



In this battle we have Explorer, who moves “SE” every time, and Line Dancer, who has two rows of Squares that replicate on the ends of the line, growing the line, while the ones in the middle attack “N” if they are in the upper row and “S” if they are in the lower row. Because they are attacking every round, any Square that comes next to them is killed before they know what hit them (unless the line dancers took more than the allotted 1ms in which case someone could sneak in and attack them while they are penalized)




Here we have Hesitant Wolf and Wanderer. Knowing where to go and not killing yourself are keys to having an awesome Team!

Wanderer moves in a random direction, looking for enemies, and replicates every once in a while. They all replicate at the same time which is fun to watch, unless they are wiping your Team out. Then it is not fun to watch.




Here is a battle of epic proportions between a Team called the Borg and itself many times over on a bigger playing field. The Borg is a fun team that would be disqualified from any self-respecting tournament because it cheats and replicates using a random class from the Squares folder. Looks like some of them replicated into the Expand class, a couple of Line Dancers, some of the Wolves, etc. This is an example of cheating… but it is a good Team to pit yours against because you never know what you will encounter! You may end up fighting yourself (:

## Getting Started

OK, you have decided on your plan of action and are ready to implement it… Here are some more detailed instructions.

1. Go to www.endigit.com/SquareBattle.
2. Download the SquareBattle Source code.
3. In the LabVIEW project you will find some folders containing example Squares, a folder of utility vis including the arena vis, the class "Square.lvclass" and then finally SquareBattle Main.vi, which when run launches the arena. There are two Build Specifications at the bottom of the project. The first, “SquareBattle”, builds the executable and you don’t need to worry about. The second, “Class Source Distribution”, you will need to right-click and select “build” after creating your Teams in order for your Starting Players to appear in the list when you run SquareBattle.exe (the executable).
4. Try it out… In the Builds folder, run “SquareBattle.exe”. Select the Squares to compete and click Start Battle. The Squares will duke it out until there's only one left or the timer runs out.

Creating your own square

1. To create your own Square competitor it must inherit from Square.lvclass (after you open the project, right click on "My Computer"->New->Class to create a class, then change the inheritance by right clicking on the class, going to properties->inheritance->"Change Inheritance" and then select "Square.lvclass") and it must reside in the "Squares" auto-populated folder. Also, it must override "Run.vi" (Right click on the class after it is set up to inherit from the Square.lvclass, choose new->"vi for override" and select "Run.vi").
2. In order for your squares to show up in the list of competitors you also must override "Player Info.vi" which is called once when the arena is launched and allows you to name your Square competitor as well as specify a victory phrase to be displayed when you win.
3. The Run.vi method of each square on the board will be called once every Round once the game starts. After all of the "Run.vi" data has been collected, the arena will be updated. The Round time is currently user adjustable during the game.
4. The Run.vi has as an input the View of what is around the square. Your job is to take that array, decide what to do, and then respond with an Action and a Direction to take that action in (as well as a Class if you are replicating). I know I've already talked about this, but I'm recapping for those that skipped the nuts and bolts section above!
5. The available actions are Wait, Move, Attack, or Replicate. Wait, Move and Attack each take one turn, Replicate takes 100 turns. When you Replicate you can pick the class of your new square. So, for example, you could have an initial square that just replicates, but when it replicates, the new squares are squares that move around and look for enemies.

## Additional Comments

- For examples of what the "Player Info.vi" and "Run.vi" should look like, look at any of the example squares in the "Squares" folder. If a class doesn't override the "Player Info.vi" you can still choose to launch that file when you Replicate (see below) but it won't show up in the list of initial players.
- To start out there are several example classes to pick from that can do battle. These are all within the Square folder.
- As soon as your class is created and "Player Info.vi" and "Run.vi" are overridden, your Square will show up in the list of players when you run SquareBattle Main.vi directly. In order for your squares to show up in the executable, you will need to run the "Class Source Distribution" build specification.
- If you click and drag on the Arena you can move it around. Feel free to center your Team when they are winning or push them off into a corner as punishment when they are losing.

So, let's see what you can do and remember to have fun!



## Who do I talk to?

- shawn@endigit.com, carl@endigit.com or robert@endigit.com