Domains

By Jon “DarkJaslo”

This document might be outdated with respect to README.md in the repository.

This game is all about painting and expanding. From two to four programs (players) compete on a board simultaneously for a number of rounds, and the player that gets the most points wins. **WARNING!** This is still a Beta, so expect some mistakes here and there.

All of this is heavily inspired in a game/competition I took part in at university. I extracted the factory pattern from there and made a similar board and core mechanics, with enough changes of my liking to make something different and more challenging while learning from the experience.

Of course, the game is functional and the documentation is there, but I don't necessarily expect nor need for it to be used. I made it just because I wanted to learn while making an interesting project. If anyone ends up making a player for it, feel free to contact me about that, it will be fun! Other of the main reasons I made this was so that I could try a lot more stuff making players for it, after all.

Contents

Mechanics

Participants code their own players, following a certain set of rules that will be specified later. To really understand everything, I recommend watching games (Game viewer).
Initially, all players are placed in a different corner of the board, with a starting domain of 3x3 squares. The objective is to accumulate points by possessing squares, popping bubbles and killing units. A variable but eventually limited number of units belong to every player, and the player must give orders to them every round. There are three possible orders: moving, attacking and using the ability.

Moving [move()]

The unit moves one Square in a direction. Valid directions include left, right, up and down. Units standing on Squares of their color can also move in diagonal directions: up-left, up-right, down-left and down-right. It is possible to order not to move by moving in direction `null`. Moving can cause the following events:

1. Drawing: the unit starts a drawing when it exits an owned area with a non-diagonal move. Moving continues the trail until it is forcefully erased or it enters an ally square again. When the latter happens, the zone bounded by the drawing is painted. This is the main way of acquiring new squares. Stepping on any drawing erases it, including ally units and own drawings. Painting a drawn square also erases its drawing.

2. Attacking: if the target position contains another unit or a bubble, it triggers a fight (more on that later).

3. Taking a bonus. If the target position contains a bonus, the unit takes it, assuming all conditions are met (also more on that later).

Attacking [attack()]

The unit attacks an adjacent position. In this case, it does not move, but it offers higher versatility: Imagine a unit u1 is in square s. Our unit, u2, wants to attack that position. If u1 moves before u2 attacks, leaving the attacked position, the move order attack would be useless. This attack order can still attack u1 if it moves but stays in attack range. This is usual when units are placed diagonally and the attacker has access to diagonal range, as could happen, for example, when defending a border. This order can be useful too if no one is in range before the attack, but a unit enters the up to three-position attack range later, which triggers a fight at the end of the round.

Abilities [ability()]

To use the ability, the unit needs to have collected a bonus beforehand, which causes it to be upgraded. Upgraded units can stay upgraded for as long as they want, considering that a team -or player- can only have one upgraded unit at a time. Using the ability generates a 5x5 painted zone, centered at the unit's position, which blocks all enemy entities from entering and exiting for a few rounds.

The ability can trigger extra painting processes if ally drawings are inside the 5x5 zone or erase them if they are from another player. If two overlapping abilities are used in the same round, they are cancelled (and thus, not used). Using an ability on top of an ability that has not worn off yet cancels its effects and applies the current ability's.

Painting

Painting events always happen because a drawing and already painted squares form a closed perimeter. This can happen when a unit that is drawing walks into a painted square, or when a bubble pop or an ability close this perimeter. For painting to happen, at least one of the perimeter's squares must be a drawing. Diagonally adjacent squares don't form a perimeter: they must be adjacent using the basic four directions: up, down, left and right.

Energy

Each unit has an energy value. At the end of each round, all units in ally squares gain 1 point and units in enemy squares lose 1 point. Anything else yields nothing. Keep this mechanic in mind with fights, because it's important there. Details on the starting and max values are in the standard configuration in *config.cnf*.

Fights

Units can fight when they are close enough. There are only two ways to trigger a fight: attacking a position with a unit on it or moving to it. Winning a fight gives points and kills the losing unit, but there is a certain risk to it:

Fair fights: they happen when both units are in strike range (eg. completely adjacent, or in diagonal but each one in owned squares). The attacker does not matter: a random number between 0 and the current energy of both units is generated for each one, and the unit that rolls the highest wins. This behaviour is very random, but gives an edge to the unit with higher energy. Also, the winning unit loses a random number of energy points in range [0,e/2], where e is the energy the losing unit had. Of course, this doesn't make its energy go below 0.

Unfair fights (or kills): they happen when the attacking unit strikes a unit that cannot counterattack because it doesn't have enough range. In this case, the striking unit always wins.

Respawns

Every three rounds (counted separately for each player), units of player p can respawn if p does not have the maximum number of units yet. For a respawn to happen, there must be at least one empty square owned by that player, and respawn locations must fit these criteria, but are chosen at random. Technically, a player cannot play anymore if they lose all squares and units.

Bubbles

Bubbles spawn in empty painted squares every three rounds, also counted separately for each player, and spawn following the same rules as units. Bubbles have a color, defined by the color of the square they spawn in. If a unit of that same color attacks it (either by moving or attacking), it pops, generating five painted squares with the pattern of a cross. If a unit of a different color does it, though, the bubble changes color to match the unit's. If nothing more is done, it pops after three rounds. Attacking it again pops it, and it being attacked by a different unit of a different color activates this process again, restarting the counter. Bubble pops give points to the player responsible for them.

Bonuses

Bonuses spawn randomly in any square of the map without a fixed frequency. Up to four bonuses can be on the board at a time. When a unit collects a bonus, it becomes upgraded, which enables the use of the ability for that unit. A player can only have one upgraded unit at a time: collecting a second bonus removes it from the board, but gives no upgrade.

Making players

Players have to inherit from the Player class, which contains the virtual method play() (don't worry, there is a player template that serves as an example). When a round begins, the game will call this function for every player, so orders to all units should be given there. Players can give up to one command per unit, and they will be executed in the same order they are given in. The order between players, though, is completely random: each time a command is going to be executed, the game decides randomly which player's it is.

You can find info on how to use the game's API in DomainsAPI.pdf.

Additional considerations

Demo

The source code provides PlayerDemo and Dummy players that can be used for demo purposes or player testing. Dummy is not intended to be good at the game, but does some stuff. As the game has many mechanics that are difficult to understand, watching sample games can serve as a starting point. Beware as Dummy never uses attack().

Openings

A good approach for any player is to design a sequence of opening moves. The beggining of the game is a very particular moment where each player has very few options, and units or bubbles will spawn every three rounds. Account for the randomness of spawn events as well as the very limited domain you start with: it will be full of units before other players end their opening sequences.

Configuration file (*config.cnf*)

Contains parameters for the game. It is local to every copy of this repository, but the version that is intented to be canon is this repository's. Please mind that I may -very rarely- tune some of the numbers in there as time goes without updating the rules here, so the definitive values will always be the ones there.

Future players

No player is currently being made, and I cannot guarantee 100% that I will even make one. However, I would like to, as the rules of this game allow for many levels of players (and let's be honest, Dummy is easy to beat).

Game viewer

The viewer program can be used to watch games. So far, it is a C++ application that runs on Linux. It takes the standard output of a Game as input, and allows moving across the rounds freely, both forwards and backwards. Space toggles the animation and left/right arrow keys allow moving from round to round.

There is also a web page with very similar capabilities, but it is still WIP.