BCOM - X-COM-like turn-based strategy

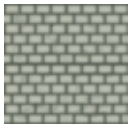
BCOM is a XCOM-like tactical game developed via Bloc in the Pharo language. You take a role of a BCOM squad commander protecting Earth from the alien invasion, and your mission is to engage and kill all alien invaders you meet on the 2D-map. The rules are simple: every soldier has 2 AP (Action Points) which can be used to perform actions. Actions can be:

- Move
- Reload
- Shoot at target

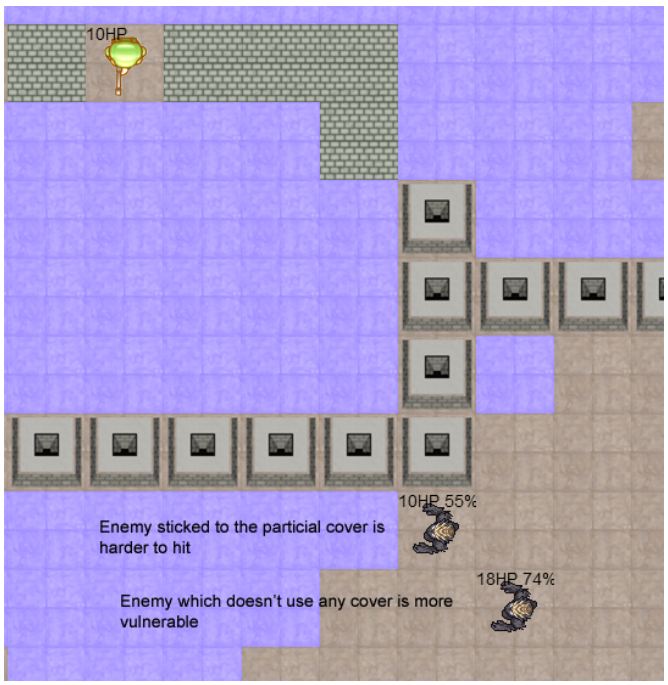
Every action costs 1 AP (except Sniper's shoot, it requires 2 AP, so you can't move/reload and shoot at the same turn). After soldier spends all his AP or fires, he ends his turn and player takes control over the next soldier he has. The player's turn ends when all player's soldiers ended their turns, and after that goes alien turn. The key to victory is a smart usage of map's landscape. Tile types are:

- concrete and grass tiles that doesn't interfere shooting and movement

- concrete and dirt partial covers that reduce hit chance to the target behind by 20%, but do not affect movement



- brick wall that works like full cover for the game character sticked to, reduces hit chance by 40% and doesn't allow to pass and shoot through



Every game character which is near the partial/full cover sticks to it and gets protection bonus in size of 20/40 percent. This bonus will be substracted from enemy's basic hit chance. Hit chance is the parameter in percents that illustrates probability of hitting an enemy with the shoot.

To move, just click the highlighted tile, and click the enemy to shoot. Remember, if hit chance is 0%, your solder will not fire!

BCOM soldiers are divided into classes, and soldier's class affects his gameplay.
- Heavy: absorbs more damage, carries assault rifle with medium stats, medium speed.
- Assault: bigger HP, moves quite fast, designed for close combat. Carries shotgun with negative precision modifier, but gets bigger precision bonus on close distances and has bigger crit and armor pierce chances.
- Sniper: low HP, slow movement, but high damage and x2 shoot range with slower hit chance decrease with range.
- Support: medium HP, nice precision and very fast movement. Carries carabine with a bit lower damage stats compared to assault rifle

While firing, weapon can generate some effects with some chance. Effects are:
- Armor pierce: target can have armor, which absorbs damage in dependence of how thick it is. Armor pierce reduces target armor by 1.
- Critical damage: bigger damage dealed by the weapon.

Installation: you need to have Bloc installed. Download source code and pictures archive, filein the codes and extract pictures into the Pharo's VM folder.

Game launch: write and doit in the Playground:

```
Game new buildUp show.
```

To display code coverage, demote all game packages to one package with tag and type and doit in the Playground:

```
Aux codeCoverage.
```

Game is built using the Bloc for graphics and controls. The main target was to enhance knowledge of Pharo, Bloc, practice in using of algorithms and programming templates. There are some examples I would like to introduce:

Polymorph work with Soldiers. Soldiers are divided into several classes, which can have difference not only in stats, but in mechanics, like Hit Chance count. There are some places where we iterate Soldier's container and counting the hit chances. It doesn't matter what is the Soldier's subclass the soldier instantiate, we use the interface defined bu the superclass. Here is the example from AlienSoldier class, AI protocol:

```
getPriorityTarget: aGame
|maxHitChance targetIndex hitChance currentIndex|
currentIndex := 0.
hitChance := -1.
targetIndex := 0.
maxHitChance := -1.
aGame human units do: [ :human | currentIndex := currentIndex + 1. hitChance := self hitChanceTo: human onMap: aGame map.
   hitChance > maxHitChance ifTrue: [ maxHitChance := hitChance. targetIndex := currentIndex. ] ].
^ targetIndex.
```

We use the Soldier's interface to get information from different Soldier's subclasses (polymorph example), and some of the Soldier's subclasses(Assault and Sniper) have overriden method hitChanceTo:onMap: which is dynamically selected while iterating(multiple dispatch).

There are a lot of composite objects in the project. Typical example of such a "has-a" relation is a "Game has a Map instance". Map instance is created and should exist while exist a Game instance. Example from Game's buildUp initializing method:

```
buildUp
    map := Map new read: self.
```

Game ending we decided to make using Visitor pattern to enhance our templates knowledge. EndGame class provides additional functionality for Game without adding any code to it: