Рефакторирање Battleships

# Battleships.ships

Template Pattern – функциите со променлива intactSegments (единици за должината на брод) се во абстрактна класа, додека децата се одговорни за поставување на должината. Додека shipGridValue е hook функцијата која се користи како генератор за уникатен број на Grid.

Strategy Pattern – избришани се две имплементации (Вертикална & Хоризонтална) и сега има само една која користи стратегиски шаблон во ShipGridSetterBehavior. Притоа се употребува Factory метода со цел да се олесни менувањето на поставеноста на бродовите (доколку во иднината се поставуваат дијагонално). Овозможува намалување на зависноста измеѓу Grid и Ship со цел во иднина контролата на поставување на бродовите да е класата измеѓу Grid и Ship.

# Enums

Додадени три нови класи.

* GridValue – енумарација на сите можни вредности што беа поставени од креаторот
* Orientation – поставеноста на бродовите на Grid
* Position – многу често има пренесување на три параметри (X,Y,Orientation) што наведува на размислување дека недостасува структура како аргумент
* ShotType – поради отсуство на типот на брод кој што се наоѓа на одредено поле во Grid, потребна беше соодветна структура која ќе посочи при удар за каков тип станува збор.

# Battleships.graphics

Template Pattern – наместо одржување на две класи со нивните имплементации за цртање, се овозможува сите бродови да одредуваат какво ќе биде цртањето во нивната супер класа, Ship. Притоа функциите на цртање е предадено на нивните деца.

# Battleships.factories

Овде овозможуваме создавањето на објектите низ апликацијата да биде поедноставно.

* GridBehaviorSetterFactory – создавање на соодветниот Strategy шаблон при поставување на бродивите на Grid
* GUIFactory – контролата на создавање на на изгледот на апликацијата е предаден на оваа класа која прима GUI-то на кое ќе го извршува цртањето. Притоа се избегнува Static дефинирање поради многубројни GUI-а кои корисникот сигурно ќе ги посака.
* ResolverFactory – овозможува намалување зависност на одредување во кој квадрант кликнал корисникот како и кој е следниот брод кој ќе биде поставен. Инаку контролата ќе беше предадена на MouseAdapter-от кој ниту малку не смее да биде зависен доколку во иднина направиме неколку нови адаптери чии логики нема да зависат од претходниот.
* ShipFactory – одговорен за креирање на нови објекти од класата Ship. Поради отсуство на Reified Generics во Java 7 (иако постоеше предлог) зависиме од класата на секој објект чие својство getSimpleName() зависи од името на соодветната класа. Истото би било возможно со Reflection на класи притоа тесно врзувајќи ги модулите со нивните класи што прилично не оневозможува да ги менуваме со други модули.
* ShipGraphicFactory – слично како ShipFactory со единствената разлика дека оваа класа е одговорна за кој Battleships.Graphics.Ship ќе биде креиран
* ShotFactory – како резултат на постоење на логиката на Agent-от потребно е да извршиме креирање на “логика” која ќе биде одговорна за одредување на следниот удар на нашиот Agent. Во зависност од бројот на hotspots ќе дефинираме Strategy шаблон. Исто така се употребува при одредување за каков брод окупира одредено поле и ажурирање кога корисникот ќе кликне на одредено поле.

# battleships.behaviors

Ги одредува сите шаблони одговорни за поставување на соодветната логика во нашата апликација.

* HShipGridSetter & VShipGridSetter – како што името потенцира, овој Template шаблон се користи за поставување на бродот на Grid во зависност на неговата ориентација. Притоа мораме да назначиме дека иако соодветните функции се прилично исти, избегнуваме од една дефиниција како резултат на промени чии директно ќе ни бидат врзани со координатите. Овие две имплементации изведуваат од ShipGridSetterBehavior чиј “рецепт” е placeShipOnGrid(int CoordinateX, int CoordinateY).
* ZeroHotspotsBehavior & OneHotspotsBehavior & ManyHotspotsBehavior – се деца на родителот-класа ShooterBehavior каде се користи Strategy шаблон за дефинирање на следниот удар на нашиот агент.

# battleships.adapters

Контролата на копчињата во нашето GUI ја превземаат следните адаптери од кои најважни се:

* AttackMousePressAdapter – каде се употребува ResolverFactory за одредување на квадрантот при клик на корисникот.
* HomeMousePressListener – каде се употребува ResolverFactory за следниот брод како и креирање на нив преку ShipFactory.

# Battleships

Најголемиот дел на рефакторирање се реализираше во овие класи чии промени беа со цел да намали зависноста меѓу соодветните објекти. Како резултат употребив MVC v1.0 каде Controller-от ја доби задачата на BattleshipsEngine, View-то ја доби задачата на GUI чиј шаблон е Composite шаблонот како резултат на компонентите. Додека Model-от ја доби задачата на GameState-от кој поради голема зависност со View-то доведе до заклучокот да се избегне MVC v2.0 чии шаблони како на пример, Observer кој ќе пренесува голем број пораки преку BattleshipsEngine, ќе ја намалат ефикасноста на апликацијата додека во исто време зголемувајќи ја употребата на меморија.

* Agent – тука употребив Strategy шаблон за одредување на следниот удар на нашиот Grid како и употреба на ResolverFactory за поставување на бродовите.
* BattleshipsEngine – иако на почеток не постоеше логика во оваа класа, кодот што постоеше во GameState за фазите во играта ги преместив овде поради MVC структурата.
* GameState – моделот во апликацијата кој на почетокот ја содржеше целата логика во апликацијата, но подоцна беше променет со цел да се проверуваат податоците на апликацијата. Затоа функциите во оваа класа се одговорни за состојбата, наспроти GUI и етапите во играта т.е. состојбите на играта.
* Grid – кој често е во интеракција со GameState (иако има неколку пристапи од GUI што ги прекршува правилата) кој освен што користи Factory шаблон за создавање на бродовите исто така ги проверува дозволувањата на матрицата.
* GUI – користи Composite шаблон со цел да биде полесно за пристап до елементите. Притоа користи GUIFactory за да го одреди изгледот на апликацијата како и GameState со цел да може да пристапи до состојбата на апликацијата и да се изменува во зависност од соодветната состојба.
* InfluenceMap – претставува AI на нашиот Agent. Никакви измени не се направени овде поради обемноста на функциите со матрицата.