Projekt na przedmiot Wzorce Projektowe

Łukasz Niemczyk

Temat: Prosta gra turowa

Język: Python m.in. biblioteka pygame

Wstęp:

Głównym celem projektu było zaprojektowanie i zaimplementowanie klas obsługujących fundamentalne mechaniki występujące w grach turowych. Program miał pozwalać na podejmowanie akcji przez wcześniej ustalone postaci (podzielone na dwie drużyny, sojusznicy kontrolowani przez gracza i przeciwnicy kontrolowani przez komputer). Wspomniane postaci są najbardziej złożoną klasą całego projektu, ponieważ to na nich opiera się cała gra. Bohaterowie mają do dyspozycji różne kategorie umiejętności, które same mogą mieć wiele rodzai. Z powodu ograniczonego czasu i tematyki przedmiotu, kreatywna i artystyczna strona nie były priorytetem, jednak kod jest przystępny do wprowadzania zmian w tych kategoriach.

Zadanie postawione przed użytkownikiem jest proste: wygrać poprzez pokonanie wszystkich przeciwników. Podczas gry, wszystkie znajdujące się na ekranie postaci (zaczynając od sojuszników) będą mogły podjąć jedną akcję w swojej turze i pozwolić zrobić to samo następnej postaci w kolejce. Sojusznicy w swoich rundach będą czekali aż gracz wybierze jaką akcję mają podjąć (a w niektórych przypadkach również postać będącą celem tej akcji), zaś za przeciwników będzie decydował algorytm. Następnie wszyscy zrobią to ponownie i ponownie, dopóki któraś ze stron nie będzie miała postaci zdolnych do dalszej walki.

Technologia:

Pygame jest darmową oper-source'owa biblioteką skupioną na narzędziach do rozwoju aplikacji multimedialnych takich jak gry komputerowe. Dlatego szybko został wybrany jako idealne narzędzie do realizacji tematu projektu. Narzędzia użyte podczas tworzenia aplikacji to m.in.: generowanie okna z wyświetlaczem ,ładowanie czcionek i generowanie z ich pomocą napisów na ekranie, sprite'y czyli małe, proste obrazy wyświetlane z zamysłem krótkiej ekspozycji, kontrola klatek na sekundę wyświetlanego obrazu, transformowanie obrazów np. odbicie lustrzane i skalowanie do odpowiednich wymiarów.

Główne klasy:

1. ### Character z adapterem Enemy

   Klasa Character obsługuje wszystkie informacje o stanie danej postaci: jej animacje, pozycję na ekranie, dostępne umiejętności, statystyki, nazwę, ale też pomoce do obsługi technicznej jak fabryka ulotnych sprite'ów informających o zmianie punktów życia czy zajmowany przez siebie prostokąt pozwalający wybrać ją jako cel.

   Konieczne było zaimplementowanie adaptera dla przeciwników, pomimo ich podobieństwa występują pewne różnice m.in. zwrócenie klatek animacji w przeciwną

stronę czy metoda pozwalająca komputerowi na zdecydowanie o podjęciu akcji bez informacji od gracza.

2. Ability

Umiejętności służą do zgrupowania zachowań często podejmowanych przez wiele różnych postaci. Dlatego ich implementacja jest na zewnątrz głównej klasy postaci (wzorzec „most”). Klasy te przechowują informację o tym czym będzie skutkowało podjęcie przez daną postać przypisanej akcji i jakie są wymagania do jej zrealizowania (np. wybranie celu jak w przypadku ataków) oraz jakim obrazem ma być symbolizowana na panelu wyboru. Pojedyncze kategorie umiejętności (np. atak w jeden cel, uleczenie samej siebie) dzielą te same schematy działania różniące się np. wyłącznie sposobem liczenia jednej wartości. Z tego powodu użyto wzorca „template method” pozwalającego zaoszczędzić czas i zmieniać jedynie wyliczanie wspomnianej wartości.

3. Healthbar

Paski życia są typowymi przedstawicielami wzorca „obserwator”. Nie są bezpośrednią częścią klasy Character, ale służą do wyświetlania informacji o jej aktualnym stanie w postaci tekstowej oraz graficznej (nazwie postaci i tego jak długo będzie w stanie dalej walczyć).

4. Module

Moduły służą do kontrolowania w jakiej części aplikacji znajduje się obecnie użytkownik. Łączy je ze sobą wyłącznie metoda run(), która obsługuje pętlę, którą przerywa decyzja użytkownika o wyjściu z aplikacji bądź zmiana modułu (np. przejście z głównego menu do rozgrywki.

5. Button

Przycisk, jak bardzo fundamentalnym elementem interfejsu by się nie wydawał, nie jest częścią biblioteki pygame. Ręcznie zaprogramowana klasa Button podczas rysowania swojego obrazu na ekranie, jednocześnie zwraca czy w danej klatce została kliknięta lewym przyciskiem myszy, co zostało użyte w wielu elementach aplikacji. Na tyle często by zasłużyć na własną fabrykę zwracającą przyciski z tekstem dopasowanym do wymiarów i ujednoliconym tłem.

6. Action Panel

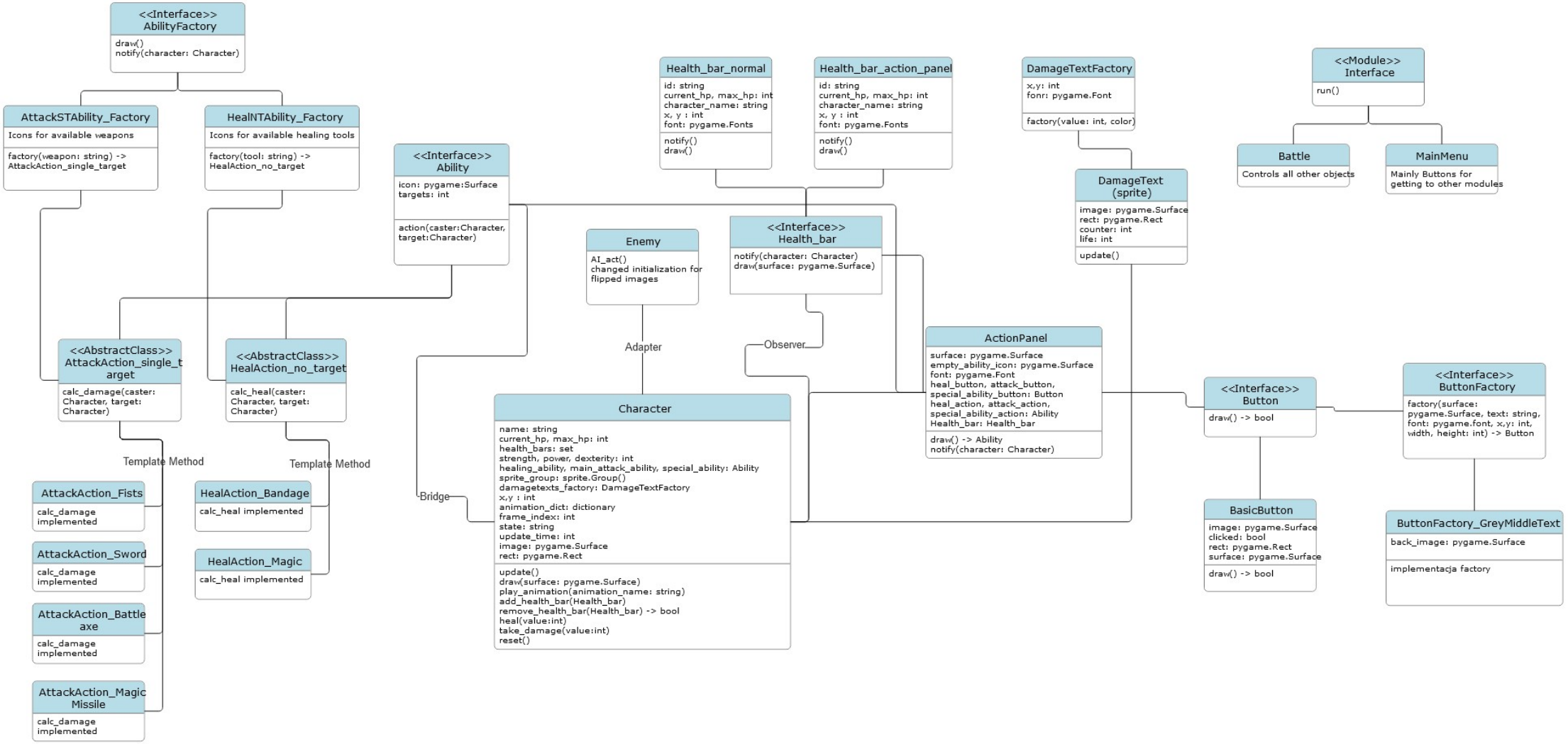
Panel wyboru akcji zostaje wyświetlany na ekranie, kiedy wymagane jest od gracza, aby zdecydował co mają zrobić postacie przez niego kontrolowane. Zawiera informacje o nazwie aktywnej postaci, jej stanie punktów życia oraz dostępnych do wyboru umiejętnościach. Panel, pomimo że blisko mu do obserwatora, nie jest przypisywany bezpośrednio do klasy Character, Panel zostaje poinformowany o zmianie aktywnej postaci przez główny Moduł, kontrolujący kolejność inicjatywy.

Prezentacja:

Wszystkie grafiki użyte w aplikacji został wykonane przez autora z pomocą internetowych narzędzi do tworzenia pixel artów, część z nich jest prosta i są skupione na przekazaniu informacji, ponieważ wykonanie większej ilości dokładniejszych grafik pochłonęłoby zbyt wiele czasu i nie byłoby związane z tematyką przedmiotu.

Z powodu surowego stanu graficznego, podczas gry, w lewym górnym rogu wyświetlona jest informacja o postaci aktualnie podejmującej działania i tym czy została wybrana umiejętność wymagająca dalszych działań ze strony gracza.

W kodzie zawarty został również tryb deweloperski, wyświetlający obszary przeznaczone na dane obiekty, który zarówno był przydatny podczas tworzenia aplikacji, ale również będzie w przyszłości pomagał w wykrywaniu błędów i dalszym rozwoju gry.


<<Interface>>
AbilityFactory
draw()
notify(character: Character)
AttackSTAbility_Factory
Icons for available weapons
factory(weapon: string) ->
AttackAction_single_target
HealNTAbility_Factory
Icons for available healing tools
factory(tool: string) ->
HealAction_no_target
<<Interface>>
Ability
icon: pygame:Surface
targets: int
action(caster:Character,
target:Character)
Health_bar_normal
id: string
current_hp, max_hp: int
character_name: string
x, y : int
font: pygame.Fonts
notify()
draw()
Health_bar_action_panel
id: string
current_hp, max_hp: int
character_name: string
x, y : int
font: pygame.Fonts
notify()
draw()
DamageTextFactory
x,y: int
fonr: pygame.Font
factory(value: int, color)
<<Module>>
Interface
run()
Battle
Controls all other objects
MainMenu
Mainly Buttons for
getting to other modules
DamageText
(sprite)
image: pygame.Surface
rect: pygame.Rect
counter: int
life: int
update()
Enemy
AI_act()
changed initialization for
flipped images
<<Interface>>
Health_bar
notify(character: Character)
draw(surface: pygame.Surface)
<<AbstractClass>>
AttackAction_single_t
arget
calc_damage(caster:
Character, target:
Character)
<<AbstractClass>>
HealAction_no_target
calc_heal(caster:
Character, target:
Character)
Adapter
Observer
ActionPanel
surface: pygame.Surface
empty_ability_icon: pygame.Surface
font: pygame.Font
heal_button, attack_button,
special_ability_button: Button
heal_action, attack_action,
special_ability_action: Ability
Health_bar: Health_bar
draw() -> Ability
notify(character: Character)
Template Method
Template Method
Bridge
Character
name: string
current_hp, max_hp: int
health_bars: set
strength, power, dexterity: int
healing_ability, main_attack_ability, special_ability: Ability
sprite_group: sprite.Group()
damagetexts_factory: DamageTextFactory
x,y : int
animation_dict: dictionary
frame_index: int
state: string
update_time: int
image: pygame.Surface
rect: pygame.Rect
update()
draw(surface: pygame.Surface)
play_animation(animation_name: string)
add_health_bar(Health_bar)
remove_health_bar(Health_bar) -> bool
heal(value:int)
take_damage(value:int)
reset()
<<Interface>>
Button
draw() -> bool
<<Interface>>
ButtonFactory
factory(surface:
pygame.Surface, text: string,
font: pygame.font, x,y: int,
width, height: int) -> Button
BasicButton
image: pygame.Surface
clicked: bool
rect: pygame.Rect
surface: pygame.Surface
draw() -> bool
ButtonFactory_GreyMiddleText
back_image: pygame.Surface
implementacja factory
AttackAction_Fists
calc_damage
implemented
AttackAction_Sword
calc_damage
implemented
AttackAction_Battle
axe
calc_damage
implemented
AttackAction_Magic
Missile
calc_damage
implemented
HealAction_Bandage
calc_heal implemented
HealAction_Magic
calc_heal implemented