

Univeritatea “Dunărea de Jos” , Galați

Facultatea de Automatica, Calculatoare, Ingineria Electrică și Electronică

Specializarea Calculatoare

**PROIECT INGINERIA PROGRAMELOR**

Katana Fruit

***Realizat de:*** Tudorache Leonard Valentin

**Grupa:** 22C32B

***Galati 2021***

**Sabloanele folosite**

**Singleton** – este un sablon creational care ne ofera o clasă cu o singură instanță, avand în același timp un punct de acces global la această instanță.

Motivele alegerii:

1. Obtinem un punct de acces global la o instanță.
2. Obiectul singleton este inițializat numai atunci când este solicitat pentru prima dată.

Implementare: in acest obiect stocam scorul, nivelul, vietile jucatorului, etc.

**Abstract Factory** – este un model de design creațional care ne permite să producem familii de obiecte conexe fără a specifica clasele lor concrete.

Motivele alegerii:

1. Evita cuplarea strânsă între produsele concrete și codul clientului.
2. Se bazeaza pe principiul deschis / închis: Putem introduce noi familii de obiecte fără a defecta codul client existent.

Implementare:

Ne ofera o metoda mai flexibilia pentru creearea a noi obiecte ce apar in joc (Fructele, elementele interfetei, etc.).

**Game Loop** – este un sablon comportamental ce asigură faptul că timpul din joc progresează cu viteză egală în configurari hardware diferite.

Implementare:

Game loop actualizeaza cadrele de comportament alte tuturor obiectelor.

**Update method** – Avem o colectie de obiecte in care fiecare obiect implementează o metodă de actualizare care simulează un cadru de comportament al obiectului. La fiecare cadru, jocul actualizează fiecare obiect din colecție.

Motivul alegerii

1. Jocul are o serie de obiecte sau sisteme care trebuie să ruleze simultan.
2. Comportamentul fiecărui obiect este în mare parte independent de celelalte.
3. Obiectele trebuie simulate în timp.

**Observer** – este un sablon comportamental care ne permite să definim un mecanism de abonament pentru a notifica mai multe obiecte despre orice eveniment care se întâmplă cu obiectul pe care îl urmareste.

Motivele alegerii:

1. Principiul deschis / închis. Când o abstractizare are două aspecte, una dependentă de cealaltă. Incapsularea acestor aspecte în obiecte separate ne permite să le modificam și să le refolosim independent.
2. Când schimbarea unui obiect necesită schimbarea altora și nu știm câte obiecte trebuie schimbate.
3. Când un obiect ar trebui să poată notifica alte obiecte fără a face presupuneri despre cine sunt aceste obiecte. Cu alte cuvinte, nu dorim ca aceste obiecte să fie strâns legate.

Implementare: Trimitem o notificare, din scena, cand jocul este terminat pentru a afisa meniul de game over.

**Pachete**

A screenshot of a computer

Description automatically generated with medium confidence

**Calcularea stabilitatii unui pachet**

Ca : Afferent Couplings : Numarul claselor din afara pachetului care depind de clasele din pachet

Ce : Efferent Couplings : Numarul claselor din interiorul pachetului care depind de clasele din afara pachetului

I : Instability : (Ce ÷ (Ca+Ce))

**Assets**

**Continut:** Elementele grafice ce sunt afisate in scena (obiecte aflate in miscare, de exemplu fructele, sau elemente ale interfetei de utilizator).

**Subpachete:** HUD, Images, Models

**Stabilitate:** (0 ÷ (1+0)) = 0

**Sabloane implementate:** Abstract Factory, Update method

**Events**

**Continut:** EventManager-ul si mai multe tipuri de listen-uri.

**Stabilitate: :** (10 ÷ (13+10)) =  **0.46**

**Sabloane implementate:** Observer

**GameEngine**

**Continut:** Inima jocului, algoritmul pentru efectuarea loop-ului si clasa abstracta ce defineste obiectele ce pot fi procesate in game loop.

**Stabilitate:** (5 ÷ (7+5))  **= 0.36**

**Sabloane implementate:** Game Loop

**Menus**

**Continut:** Interfete de utilizator

**Stabilitate:** (5 ÷ (6+5))  **= 0.45**

**Scenes**

**Continut:** Scenariile jocului, fiecare scenariu poate oferii o experienta de joc complet diferita.

**Stabilitate:** (8 ÷ (7+8)) =  **0.53**

A picture containing graphical user interface

Description automatically generated

Figure 1 Diagrama cu toate clasele si dependentele dintre ele

**Application Flow**

Graphical user interface, application, Teams

Description automatically generated

Figure 2 Application flow

**Window**

Window este clasa care se ocupa cu creearea ferestrei si initializarea componentelor pentru a incepe jocul. Dupa creearea ferestrei, in aceasta, se adauga meniul principal (MainMenu) caruia ii este transimis prin constructor EventManager-ul si Container-ul JFrame-ului. EventManager-ul este initializat cu un eveniment numit “gameEnded” care are ca scop oprirea jocului si schimbarea continutului ferestrei cu meniul aferent termianarii jocului.

**A picture containing text, sign, screenshot

Description automatically generated**

**EventManager**

Este clasa care se ocupa cu stocarea unor n numar de actiuni si rularea lor atunci cand este nevoie.

**Graphical user interface

Description automatically generated with medium confidence**

**Menu**

Meniurile sunt interfata grafica care face legatura dintre utilizator si program. Acestea contin o arie de optiuni ce pot fi actionate de utilizator. In program acestea ofera o manevrare mai usoara a game loop-ului si a scenei.

**Graphical user interface, application, Teams

Description automatically generated**

**StartGameAction**

Este un action listener separat de actiunile din EventManager. Aceasta clasa are ca scop rularea instructiunilor pentru a incepe jocul.

**Graphical user interface, text, application

Description automatically generated**

**Scene**

Scene (Handler) este clasa ce are ca scop afisarea si actualizarea comportamentului obiectelor, obligatoriu de tip GameObject, adaugate in scena. Pe langa acestea, scena se mai ocupa si cu inregistrarea inputului utilizatorului. Clasa “Scene” ofera programatorului posibilitatea de a creea si manipula, individual sau intr-un numar mai mare, toate obiectele aflate in scena.

**Timeline

Description automatically generated**

**GameObject**

GameObject este o clasa abstracta care defineste metodele si proprietatile necesare pentru a putea creea un obiect ce poate fi reprezentat si procesat in scena. Orice Clasa care mosteneste clasa GameObject mosteneste niste proprietati necesare pentru reprezentarea grafica a unui obiect 2D, pe langa aceste proprietati, clasei ii se impune implementarea urmatoarelor metode:

- update() : Controleaza comportamentul obiectului.

- setTextures() : Incarca texturile necesare obiectului.

- draw() : Reprezinta graphic obiectul.

- handleInput() : Ofera posibilitatea de a controla comportamentul obiectului in funcite de input.

**Graphical user interface, text, application

Description automatically generated**

**GameLoop**

Este clasa care implementeaza sablonul cu acelasi nume, aceasta clasa porneste un thread nou in care porneste o bucla while in care se reimprospateaza scena si obiectele aflate in aceasta cat timp statusul jocului este “RUNNING”.

A picture containing timeline

Description automatically generated

**Teste unitare**

Acestea au fost facute cu ajutorul librariei JUnit, o librarie dedicate testelor unitare. In proiect avem urmatoarele teste efectuate:

* Coliziunea dintre obiecte:

1. In primul test am creeat 2 obiecte cu aceleasi proprietati si am verificat daca punctele acestora se intersecteaza.
2. In al doilea test am despartit aceste puncta cu o distanta de 99 pe axa XY pentru a verifica daca testul precedent este valid.

* Verificarea implemetarii corecte a sablonului singleton

1. Am verificat egalitatea a doua instante ale obiectului singleton, prin metoda getStatus()
2. Am verificat daca implementarea sablonului singleton este threadsafe prin pornirea unui nou thread si schimbarea unei proprietati din obiectul singleton.