Zadaća 1

Računalna igra

[Softversko inžinjerstvo]

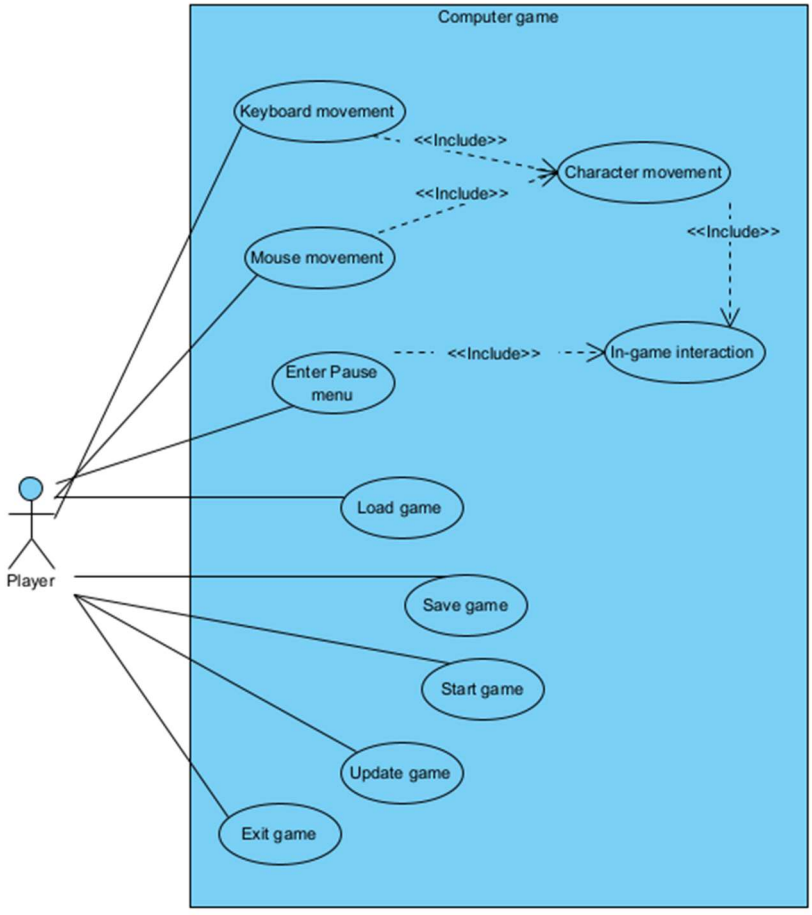
Mateo Martinjak

Zadatak:

Računalna igra simulira kretanje virtualnog junaka kroz labirint i njegovu borbu s neprijateljima. Stanje igre prikazano je na zaslonu računala kao 3D pogled na labirint iz očiju junaka. Prikaz je nadopunjen dodatnim informacijama kao što su osvojeni bodovi, "zdravlje" junaka, oružje koje junak trenutno koristi, stanje municije, itd. Igrač upravlja kretanjem i akcijama junaka uz pomoć tipkovnice i miša. Računalna igra povremeno generira virtualne neprijatelje koji napadaju junaka i koje junak treba ubiti. Igra također obračunava i bilježi bodove koje je igrač stekao svojom igrom. Podržano je zaustavljanje igre, bilježenje trenutne situacije na disk računala te ponovno učitavanje zabilježene situacije u kasnijem trenutku. Također je podržano skidanje "zakrpi" (updates) igre s web sjedišta proizvođača igre.

1. Use case dijagram

Pošto se radi o single player igri, nema pretjerano kompliciranih use caseova. Igrač je jedini akter u cijeloj priči (osim web servera preko kojeg se igra ažurira).Veća težina projekta bit će na samoj igri, dok su Update, Load i Save opcionalne mogućnosti koje će se pokriti naknadno.Ukratko, igrač ima mogućnost kretati se svijetom te napadati neprijatelje i sakupljati resurse.



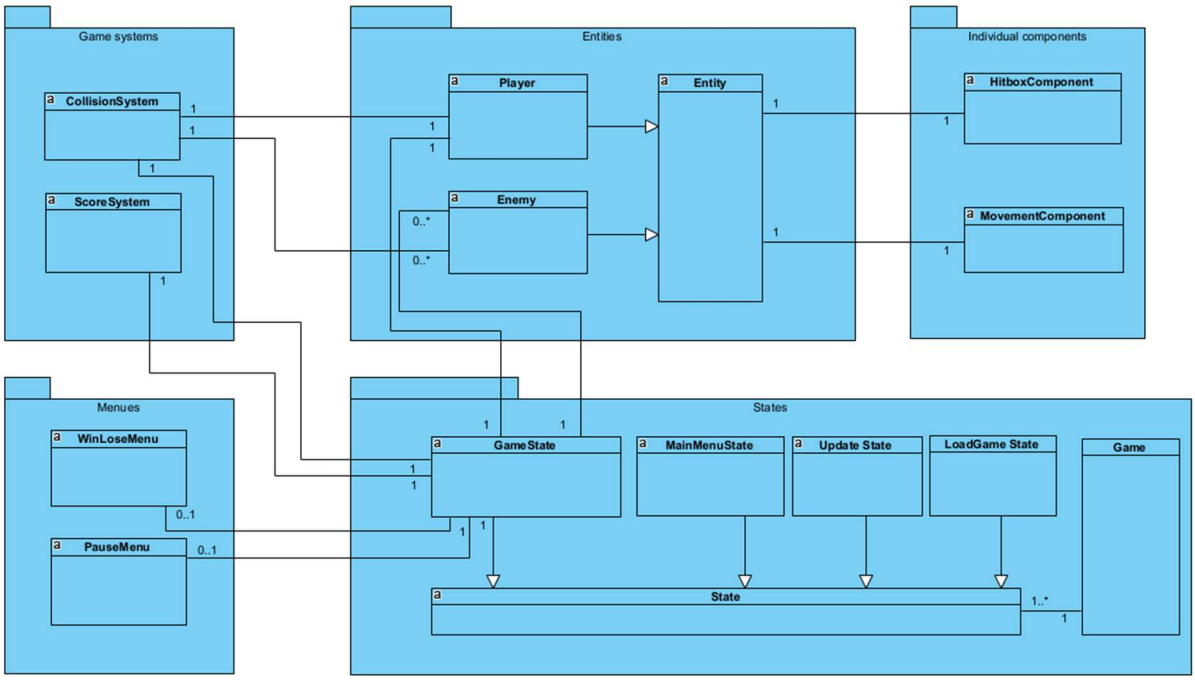
Slika 1 Use case dijagram osnovnih use caseova.

Use case: Update game
ID: 1
Opis: Igra automatski traži ažuriranja s web servera
Primarni akter: Igrač
Sekundarni akter: Server
Preduvjeti: Igra mora biti pokrenuta
Glavni tok: 1. Igra se pokrene 2. Igra pri pokretanju uspoređuje svoju verziju s verzijom na web serveru 3. Sustav daje igraču mogućnost skidanja zakrpe (ako postoji) 3.1. Ako je zakrpa sigurnosna ,igrač ju mora prihvatiti ili nemože igrati igru 3.2. Ako je zakrpa za unaprijeđenje igre, zakrpa je opcionalna
Naknadni uvjeti: Igra se mora ponovnoo pokrenuti ukoliko se instalira zakrpa.
Alternativni tokovi: nema

Use case: Enter pause menu
ID: 2
Opis: Prilikom igranja, igrač ima izbor pauzirati igru
Primarni akter: Igrač
Sekundarni akter: nema
Preduvjeti: Igra mora biti pokrenuta i učitana
Glavni tok: 1. Igra se pokrene te se učita pomoću Start game/Load game 2. Ako u tom trenutku na zaslonu nije „Win screen“ ili „Lose screen“, igrač može pritiskom na ESC pauzirati igru 3. Na ekranu se tada pojavljuje button za izlazak iz igre, button za save igre te dodatni podaci o trenutnoj sesiji igre (zdravlje igrača,municija itd)
Naknadni uvjeti: Nema
Alternativni tokovi: Nema

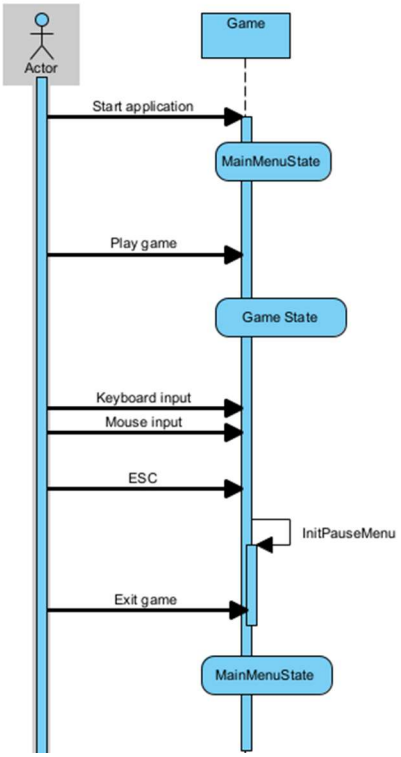
2. Class dijagram

Sljedeći dijagram opisuje ugrubo od kojih klasa se sastoji igra, niže je objašnjeno zašto su baš ovako kategorizirane klase i što koja od njih radi.



Slika 2 Class dijagram klasificiran u pakete

3. Interna logika aplikacije



Slika 3 Opis jednostavne promjene statea

3.1. State Pattern

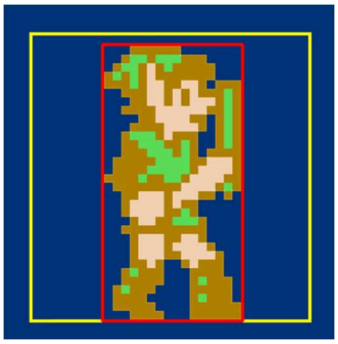
Ovaj class diagram pokazuje ugrubo od čega se projekt sastoji (niže je detaljnija specifikacija svakog elementa). Cijela igra (projekt) se bazira na „state design patternu“: igra se u svakom trenutku nalazi u nekom stateu, najvažniji stateovi su GameState i MainMenuState. Game klasa drži u sebi stog s pokazivačima na State objekte. Dakle kada se sama igra pokrene, Game inicijalizira taj stog tako da u njega ubaci MainMenuState. (prokazuje se na ekran onaj state koji je najviši na stogu) Kada se u glavnom meniju klikne na Start game, MainMenuState se ne briše sa stoga nego MainMenuState inicijalizira i ubacuje GameState na taj stog, te se u tom trenutku GameState iscrtava na ekran.

Slika 4 Stog u Game klasi: Crta se na ekran onaj state koji je najviši na stogu

3.2. GameState

Fokusirajmo se sada samo na GameState:

GameState se u radnoj petlji brine o protivnicima,te ih nasumično inicijalizira.Player se automatski inicijalizira na početku igre.Svaki Entity u igri ima svoj poseban HitboxComponent i MovementComponent. Naime kada kliknemo na tipkovnicu da bi pomakli lika (ili to napravi sustav da bi pomaknuo neprijatelja), oni zapravo to rade preko komponente MovementComponent tako da toj komponenti proslijede gdje se neprijatelj/lik želi pomaknuti, tada će ga komponenta pomaknuti koliko treba. To je napravljeno tako jer nam pruža apstrakciju oko pomicanja, jer se komponenta brine o tome hoće li lik pomakom proći kroz ekran, jel se pomiće sporo itd.. Hitbox komponenta je ustvari običan kvadar koji opisuje našeg lika/neprijatelja. Treba nam jer je >puno< lakše računati kolizije između objekata kad su obični geometrijski objekti u pitanju. Pa kad se lik zabije u zid zapravo ćemo računati s tim kvadrom udaljenosti. (taj hitbox-kvadar se očito uvijek pomiče s likom).

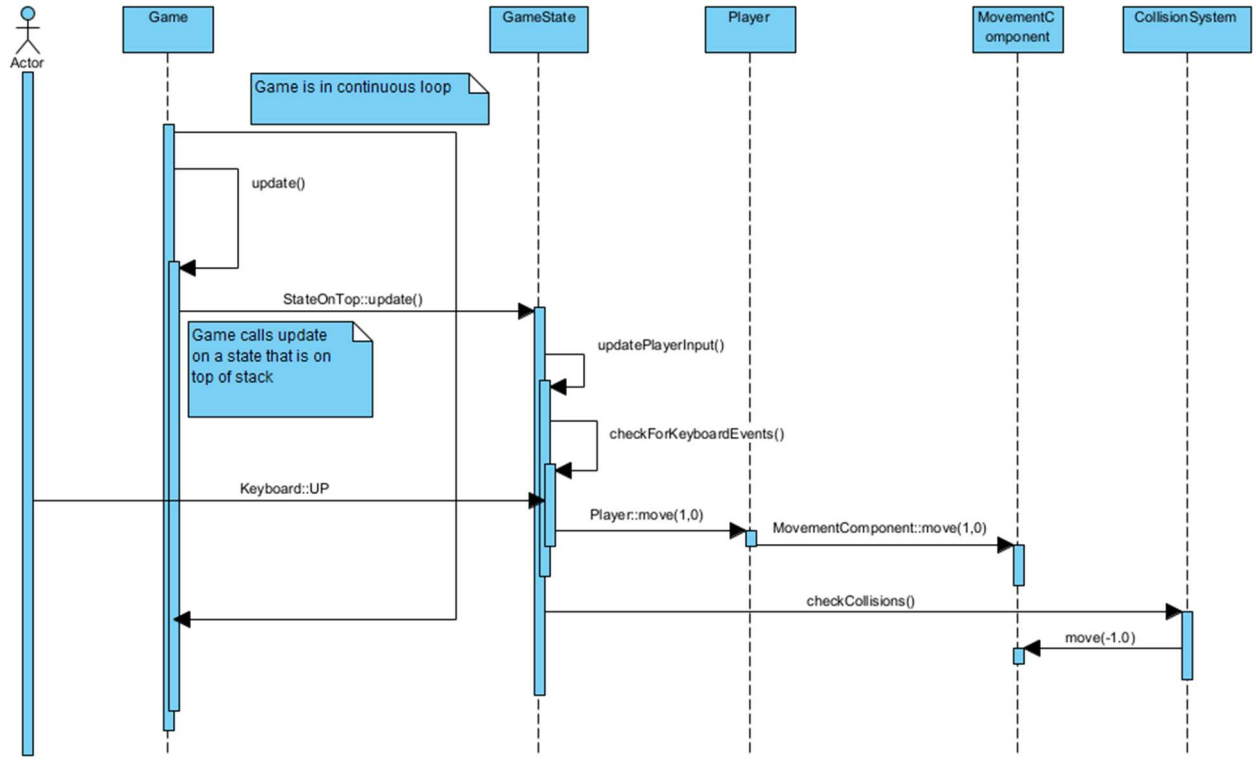


3.3. Collision i score sistemi

Nadalje imamo collision system i score system. GameState u svakoj iteraciji pozove jednom ta dva sistema da odrade svoj posao, te se tada kontrola vraća GameStateu.

Ta dva sistema imaju kao referencu cijelu scenu, tj. da bi npr. Izračunali kolizije između lika i zida, CollisionSystem mora imati u svakoj iteraciji referencu na oboje. Kolizije koje računamo su apsrahirane u tu navedenu klasu zbog jednostavnosti. (bez te klase, morali bi provjeravati kolizije u klasi GameState). ScoreSystem također ima referencu na sve da bi znao „donositi odluke“ o tome koliko bodova igrač skupit u nekom trenutku. Skupi li igrač dovoljno bodova ,ili ukoliko ga neprijatelj ubije triggera se WinLoseMenu.

WinLoseMenu i PauseMenu mogući su jedino u GameState-u. To su „polu-stanja“ (ne ulaze u onaj gore spomenuti stog), nego se stvore/unište dinamički kada ih korisnik pozove.



Slika 5 Sequence dijagram koji prikazuje kako se character pomiče u svijetu. U ovom primjeru character se pomaknuo u zid, te u istom koraku (prije nego se ta slika prikaže na ekran) CollisionSystem ga vrati mjesto gdje je bio jer je primijetio da su character i zid u koliziji (Collision system ima reference za sve objekte u sceni, bitno ga je samo pokrenti prije crtanja svakog framea)

Opis sequence dijagrama:

Game u beskonačnoj petlji updejta State koji se nalazi na vrhu stoga. GameState u svom update() provjerava jeli korisnik pritisnuo tipku. Je, znači poziva se odgovaraća fja. move(). Ali GameState nije gotov, na kraju GameState::update() također poziva i CollisionSystem koji provjerava postoji li kolizija u cijeloj sceni. Ako postoji, ta klasa ima reference na ostale objekte u sceni (uključujući našeg playera) pa mu mijenja položaj kako nebih prošao kroz zid ili kroz neprijatelja.