# Uvod

Logičke računalne igre (engl. Puzzle Video Games) igraču predstavljaju problem koji je rješiv primjenom logičkog zaključivanja nad skupom pravila. Za razliku od mnogo ostalih žanrova igara, ne zahtijevaju mehaničku spretnost ili brzinu reakcije, dakle tempo igranja im je sporiji i postupak rješavanja razine opušteniji. Postoje iznimke (npr. Tetris) gdje je vremenski faktor ključan. Povijesno gledano, logičke igre bile su među prvim video igrama koje su stekle veliku popularnost i opću prepoznatljivost. Zbog jednostavnosti dizajna, kao i činjenice da su dolazile unaprijed instalirane na mnoge operacijske sustave bile su pristupačne gotovo svakome (npr. Minesweeper ili Solitaire na Windows OS-u). Kasnije su nastajale i logičke igre smještene u 3D okruženju koje dijele svojstva s naslovima drugih žanrova poput FPS (npr. Portal). Postojeća igra koja je najviše utjecala na ovaj rad je vjerojatno Sokoban [2], japanska video igra dizajnirana 1981. u kojoj igrač slaže kutije na predviđena polja u kompaktnoj mapi koja predstavlja skladište. Pravila igre ovog rada su sljedeća: Na pravokutnu podlogu zadanih dimenzija generiraju se objekti – kutije i igračev lik – zvrk. Cilj igre je dovesti zvrk na označeno ciljno polje. Korisnik može micati zvrk ili bilo koju kutiju i to tako da ju pošalje u jednom od četiri osnovna smjera (gore, desno, dolje, lijevo) pravocrtno dok se ne zabije u drugi objekt ili ne odleti s podloge. Pri uspješnom rješavanju razine, automatski se generira novi, nasumično postavljajući kutije, no osiguravajući rješivost. Također je moguće podesiti dimenzije podloge i broj kutija koje će stvoriti na njoj. Zbog nasumične prirode stvaranja mape, nikad nije osigurana kompleksnost, no podešavanjem parametara može se mijenjati vjerojatnost za generiranjem zahtjevnije razine. Ideja je da igra potiče rješavanje većeg broja razina u jednoj sjednici i da igrač bude zadovoljan ukupnom zahtjevnošću skupa odigranih razina.

# Algoritam za generiranje mape

Tijek igre zasniva se na konstantnom regeneriranju mape, odnosno područja za igru, koje uključuje podlogu određene veličine, igračev zvrk te određeni broj kutija i njihov raspored. Pri prvom pokretanju igre i svakim uspješnim prelaskom razine ili proizvoljnim traženjem novog poziva se funkcija koja provodi algoritam za generiranje mape na temelju zadanih ulaznih parametara i sjemena za generator nasumičnih brojeva.

## Generator mape

Razred MapGenerator, prikazan u kodu 1.1, obuhvaća sve potrebne funkcije i parametre za stvaranje rješive razine igre. Ulazni argumenti konstruktora su: širina mape, visina mape i broj linija ispravnog rješenja. Prilikom konstruiranja instance razreda kreira se matrica koja se uređuje pozivom funkcija, te se nakon prolaska kroz „cjevovod“ na izlazu dobije matrica formata prikladnog za korištenje prilikom kasnijeg generiranja 3D objekata u sceni igre.

class SPARKYPUZZLE\_API MapGenerator {

public:

MapGenerator(int width, int height, int lines);

private:

int MAP\_X;

int MAP\_Y;

int N\_OF\_LINES\_TO\_WIN;

int STARTING\_X;

int STARTING\_Y;

int ENDING\_X;

int ENDING\_Y;

public:

TArray<TArray<char>> MAP\_MATRIX;

int getRandom(int lower, int upper);

int generatePaths();

int placeBoxes();

int formatLayout();

int printLog();

};

Kod 1.1 – Razred MapGenerator

## Stvaranje putanje

Algoritam osigurava rješivost razine upravo unutar funkcije generatePaths i to na način da je prvo što se određuje putanja od početne lokacije zvrka do ciljnog polja. Prije poziva ove funkcije matrica mape je prazna, a prvo što funkcija u nju upiše su nasumično određene početna (S) i završna (Z) pozicija. Nakon toga zadani broj puta (pozivom konstruktora) odredi smjer i duljinu sljedećeg pomaka na putanji (P) koja vodi od početne do završne pozicije. Na posljetku na kraj svake putanje postavi kutiju (B) te je time stvorena mapa s kutijama postavljenim na način da zvrk može proći od početka do kraja.

Prilikom određivanja smjera kretanja, treba spriječiti ponavljanje istog smjera, besmisleno vraćanje istim putem i pad s ruba mape te riješiti poseban slučaj kada je putanja na dva ili manje koraka do kraja. To se postiže uređenjem liste dostupnih smjerova spremljene u varijablu available\_directions, odnosno izvršenjem dijela programa prikazanog u kodu 1.2.

n\_of\_possible\_directions = 0;

for (int i = 0; i < 4; i++) {

if (available\_directions[i] == 1) n\_of\_possible\_directions++;

}

if (n\_of\_possible\_directions == 0) return 1;

previous\_direction = current\_direction;

current\_direction = getRandom(0, n\_of\_possible\_directions - 1);

n\_of\_found\_directions = 0;

for (int i = 0; i < 4; i++) {

if (available\_directions[i] == 1) {

if (current\_direction == n\_of\_found\_directions) {

current\_direction = i;

break;

}

else {

n\_of\_found\_directions++;

}

}

}

Kod 1.2 – određivanje smjera linije putanje

Napominjem da je lista dostupnih smjerova osmišljena kao četveročlana lista gdje indeks predstavlja smjer (0 – lijevo, 1 – gore, 2 – desno i 3 – dolje), a vrijednost predstavlja dostupnost smjera (0 – nedostupan, 1 – dostupan).

Nakon smjera, određuje se duljina linije putanje, a na kodu 1.3 prikazan je taj postupak za smjer kretanja u lijevo.

if (current\_direction == ((previous\_direction + 2) % 4)) min\_length = length / 2;

else min\_length = 1;

length = 0;

loop\_counter = 0;

if (current\_direction == 0) {

if (line == (N\_OF\_LINES\_TO\_WIN - 2) || line == (N\_OF\_LINES\_TO\_WIN - 1)) {

length = current\_x - end\_x;

} else {

length = getRandom(min\_length, (current\_x - 1));

while (((current\_x - length == end\_x) || (current\_x - length == end\_x + 1)) && (current\_y == end\_y)) {

length = getRandom(min\_length, (current\_x - 1));

if (loop\_counter == 5) return 1;

loop\_counter++;

}

}

for (int x = 0; x < length; x++) {

current\_x -= 1;

if (MAP\_MATRIX[current\_y][current\_x] != 'B') MAP\_MATRIX[current\_y][current\_x] = 'P';

}

MAP\_MATRIX[current\_y][current\_x - 1] = 'B';

}

Kod 1.3 – Određivanje duljine linije putanje

## Postavljanje kutija

U nastojanju da uspješan prelazak razine ne bude trivijalan, unutar funkcije placeBoxes dodaju se nove kutije i već stvorene se pomiču s početnih položaja, što rasporedu objekata na razini daje privid nasumičnosti i čini da rješenje ne bude očito. To se radi iteracijom po svakom polju mape gdje ako se na pojedinom polju nalazi kutija, određuje se smjer putanje pomaka kutije na čija se oba kraja postavljaju nove „sekundarne“ kutije (b). Time se svakoj početnoj kutiji omogući jedan stupanj pomaka po pravcu. U kodu 1.4 vidi se provjera mogućnosti te postavljanje sekundarnih kutija za potencijalni smjer u lijevo.

potential\_direction = getRandom(0, 3);

if (potential\_direction == 0) {

if (current\_x == 0 || current\_x == 1 || current\_x == (MAP\_X - 1)) continue;

potential\_distance = getRandom(2, current\_x);

if (MAP\_MATRIX[current\_y][current\_x - potential\_distance] != '.') {

if (MAP\_MATRIX[current\_y][current\_x - potential\_distance] != 'b'

&& MAP\_MATRIX[current\_y][current\_x - potential\_distance] != 'B') continue;

}

if (MAP\_MATRIX[current\_y][current\_x + 1] != '.') {

if (MAP\_MATRIX[current\_y][current\_x + 1] != 'b'

&& MAP\_MATRIX[current\_y][current\_x + 1] != 'B') continue;

}

MAP\_MATRIX[current\_y][current\_x - potential\_distance] = 'b';

MAP\_MATRIX[current\_y][current\_x + 1] = 'b';

boxesPlaced = true;

MAP\_MATRIX[current\_y][current\_x - potential\_distance + 1] = 'b';

if (MAP\_MATRIX[current\_y][current\_x] != 'X') MAP\_MATRIX[current\_y][current\_x] = '.';

}

Kod 1.4 – postavljanje dodatnih kutija na mapu

## Formatiranje matrice

Nakon što su na mapu postavljeni sve potrebni objekti, potrebno je još jednom proći kroz matricu mape i pobrinuti se da su u njoj samo simboli potrebni za stvaranje 3D objekata, dakle „.“, „S“, „X“, i „B“, odnosno treba izmijeniti pomoćne simbole „P“ i „b“. Cijela funkcija formatLayout je prikazana u kodu 1.5.

for (int i = 0; i < MAP\_Y; i++) {

for (int j = 0; j < MAP\_X; j++) {

if (MAP\_MATRIX[i][j] == 'P') MAP\_MATRIX[i][j] = '.';

if (MAP\_MATRIX[i][j] == 'b') MAP\_MATRIX[i][j] = 'B';

}

}

return 0;

Kod 1.5 – funkcija za formatiranje matrice mape

Funkcija getRandom vidljiva u prethodnim primjerima je metoda razreda koja koristi funkciju rand standardne c++ knjižnice kako bi vratila nasumičan broj unutar zadanih granica:

return ((rand() % (upper - lower + 1)) + lower);

Izgled matrice mape nakon poziva generatePaths:

**. B P P P P B P P S**

**. . . . . P . . . .**

**. . B P P P . . . .**

**. . . P . B . . . .**

**. B . P . . . . . .**

**. B P P P P P X B .**

**. P P B . . . . . .**

**. P P . . . . . . .**

**B P P . . . . . . .**

**. . B . . . . . . .**

Izgled matrice nakon poziva placeBoxes:

**. B P P P P B P P S**

**. . b . . P . . . .**

**. . . P P P . . b .**

**b b . P . . b . b .**

**b . . P . . b b . .**

**. B P P P P P X . .**

**b b P . b . . . b .**

**. P P . . . . . . .**

**. P b . . . . . . .**

**b . b . . . . . . .**

Izgled matrice nakon poziva formatLayout:

**. B . . . . B . . S**

**. . B . . . . . . .**

**. . . . . . . . B .**

**B B . . . . B . B .**

**B . . . . . B B . .**

**. B . . . . . X . .**

**B B . . B . . . B .**

**. . . . . . . . . .**

**. . B . . . . . . .**

**B . B . . . . . . .**

# Korištene tehnologije i alati

Cjelokupna izvedba programskog rješenja pisana je programskim jezikom c++, unutar grafičkog pogona Unreal Engine 4.27. Prvobitni kod generatora mape razvijen je koristeći samo standardne knjižnice za c++, no pri njegovom uklapanju u UE4 projekt, bilo je nužno zamijeniti neke podatkovne strukture i metode onima koje su dio UE4 knjižnice. Modeli objekata scene izrađeni su u programu za modeliranje – Blender, a materijali su izrađeni u programu za uređivanje slika – Photohop.

## Unreal Engine 4.27

Unreal Engine je pogon za izgradnju video igara koji je razvio Epic Games i izašao je 1998. izvorno nastavši kao pogon za igru Unreal. Vrlo je opsežan i nudi mnoge funkcionalnosti korisne za izradu grafičkih aplikacija kao što je sustav osvjetljenja, sustav za efekte čestica, sustav za izradu materijala, mogućnost vizualnog programiranja (Blueprints) itd. Za izradu ovog rada korišteno je Unrealovo sučelje za podešavanje materijala, sučelje za podešavanje utjecaja svijetla, sučelje za prihvat korisničkih naredbi te opsežni radni okvir za definiranje elemenata igre i generički razredi čijim se nasljeđivanjem mogu koristiti mnoge korisne metode. Također, podržava gradnju aplikacija za razne sustave.

## Blender

Blender je program koji sadrži skup alata za izradu, uređivanje i iscrtavanje 3D modela. Sadrži mnogobrojne metode i načine za interakciju s modelom i čini njegovo uređivanje intuitivno i pristupačno. Podržava izvoz modela u raznim formatima.

## Photoshop

Photoshop je Adobeov program za uređivanje fotografija, no pruža alate za crtanje i uređivanje boje i efekata slike koji su prikladni za stvaranje jednostavnih tekstura.

# Modeli i materijali

Za izradu 3D modela za igru korišten je Blender, a napravljena su tri modela: podloga, kutija i zvrk. Cilj je bio na relativno jednostavan način uređivanjem osnovnih modela (valjak i kocka) dobiti mrežu s relativno malim brojem vrhova koja nije previše zahtjevna za iscrtati.

## Podloga

Za izradu podloge jednostavno je mreža kocke skalirana po z-osi u oblik niskog ali širokog kvadra. Slika 3.1 prikazuje iscrtani model podloge. Isti model je korišten i kao podloga za objekte i kao oznaka ciljnog polja.

A white square object on a black background

Description automatically generated

Slika 3.1 - Model podloge

## Kutija

Kutija je oblika kocke, ali je modifikatorom boolean, izdubljena sa svih strana čime se dobije istaknuti obrub kutije, nakon toga je prilikom dodatka krnjih kutova korišten modifikator mirror kojim po svim trima osima zrcalimo uradak na jednom rubu. Na posljetku su vrhovi suprotnih kutova svake strane povezani kako bi se dobila izražena dijagonala i postigao prepoznatljivi izgled drvene skladišne kutije. Slika 3.2 prikazuje iscrtani model kutije.

A white cube with a black background

Description automatically generated

Slika 3.2 - Model kutije

## Zvrk

Zvrk je kao polaznu mrežu koristio valjak te se postupcima izvlačenja i skaliranja dijelova mreže postigao klasični izgled zvrka. Kao dodatak su izvučeni stupovi na vršnom prstenu i na vrh je dodana izdužena polovica ikosaedra kako bi se dobio specifičan i svojstven izgled igračevog lika. Slika 3.3 prikazuje iscrtani model zvrka.

A grey object with a black background

Description automatically generated

Slika 3.3 - Model zvrka

## Materijali

Jednostavne teksture napravljene su u Photoshopu koristeći jednobojne kvadratne slike kao polazišta. Za efekt teksture podloge i ciljnog polja korišten je kist i kist za miješanje, za kutije alat gradijenta, a za zvrk također kist i kist za miješanje, uz dodatak kantice za popunjavanje. U Unrealovom sučelju za izradu materijala navedene teksture predane su kao osnovna boja, te je napravljen posebni materijal za svaki tip modela te posebno za kutiju i zvrk kad su odabrani mišem. Slika 3.4 prikazuje navedene teksture.

A group of different colors of different shapes

Description automatically generated

Slika 3.4 - Teksture korištene u radu po redu: podloga, ciljno polje, kutija, označena kutija, zvrk, označeni zvrk

# Struktura programskog rješenja

Kao osnova funkcionalnosti igre koristi se međudjelovanje dvaju razreda izvedenih iz generičkih funkcija koje su dio Unrealovog skupa alata za izradu video igara: GameModeBase i PlayerController. Praksa pri razvoju proizvoda koristeći Unreal je nasljediti te razrede koji sadrže razne metode korisne za upravljanje tokom igre te napraviti vlastitu implementaciju prikladnu za vlastiti proizvod. Uz ta dva upravljačka razreda, za stvaranje statičnih objekata izvode se razredi iz Actor, a za objekte kojima bi igrač trebao moći upravljati iz Pawn. Slika 4.1 prikazuje UML dijagram razreda koji odražava strukturu projekta. BirdCamera nasljeđuje Actor i enkapsulira sve komponente pogona vezane za kameru u sceni. Nakon generiranja mape stvore se dvije kamere, jedna točno iznad središta podloge koja gleda prema dolje i daje tlocrtni prikaz mape, a druga koja je sa strane i daje prikaz mape pod kutom.

A diagram of a computer program

Description automatically generated with medium confidence

Slika 4.1 - UML dijagram razreda

## Razred upravljivog objekta

Cubicle nasljeđuje Pawn i obuhvaća funkcionalnosti vezane za logičku obradu kretanja objekta što je prikazano u kodovima 4.1 i 4.2 te zamjene materijala po potrebi,. CubicleBox i CubicleSparky nasljeđuju Cubicle i sadrže konstruktor u kojem postavljaju vrijednosti mreže i materijala objekta specifične za kutiju i zvrk. Floor nasljeđuje Actor i postavlja mrežu i materijal svojstvenu podlozi i ciljnom polju.

void ACubicle::SlideMove(FVector EndLocation)

{

if (!bIsMoving) {

MoveEndLocation = EndLocation;

MoveDirection = (MoveEndLocation - GetActorLocation()).GetSafeNormal();

bIsMoving = true;

}

}

Kod 4.1 – postavljanje vrijednosti potrebnih za pomak

if (bIsMoving) {

FVector locationIncrement = GetActorLocation() + MoveDirection \* 6000.0f \* DeltaTime;

SetActorLocation(locationIncrement);

if ((GetActorLocation() - MoveEndLocation).Size() < 100.0f) {

bIsMoving = false;

SetActorLocation(MoveEndLocation);

if (MoveEndLocation.Z == 121.1f) Destroy();

}

}

Kod 4.2 – dio koda koji služi za pomicanje objekta, a nalazi se unutar funkcije Tick koja se poziva svaki okvir

# Način rada igre

Razred SparkyPuzzleGameModeBase nasljeđuje GameModeBase i u njemu je implementirana funkcionalnost generiranja matrice mape, generiranja objekata u sceni iz matrice te sadrži sve strukture i objekte kojima SparkyPlayerController pristupa.

## Stanje igre

Članska varijabla state služi kako bi se opisalo u kakvom stanju se nalazi igra, što je korisno pamtiti zato što je na temelju toga moguće blokirati ili dopustiti određene funkcije od izvršavanja.

Stanja igre koja su implementirana:

* IN\_PROGRESS – igra traje, a svi objekti stoje
* SUCCESS – zvrk stoji na ciljnom polju
* IN\_MOTION – neki objekt se kreće

## Funkcija za osvježavanje mape

Funkcija resetMap, prikazana u kodu 5.2 poziva se kada se želi generirati mapa na temelju već postavljenog sjemena, čime se može po potrebi ostvariti osvježavanje trenutne razine (npr. Ako zvrk izleti s mape). Funkcija poziva drugu funkciju – generateMap, koja poziva sve metode razreda MapGenerator i vraća spremnu matricu mape, koju koristi za poziv funkcije za stvaranje objekata scene te postavlja stanje igre na IN\_PROGRESS.

srand(seed);

MapGenerator map = generateMap(width, height, lines, objects);

generateCubicles(map);

state = 0;

Kod 5.2 – Funkcija za osvježavanje scene

## Rječnik igre

Rječnik igre - GameMap ostvaren je podatkovnom strukturom rječnika (TMap u Unrealu) kojoj je ključ string, a vrijednost boolean. Ključevi rječnika igre imaju oblik „{x}//{y}“, gdje {x} i {y} predstavljaju komponente vektora lokacije polja u sustavu mape. Vrijednost ključa je true ako se na tom polju nalazi objekt, a laž ako ne. Ti podaci se osvježavaju svakim pomakom objekta tijekom igre, a temeljni su dio računa logike pomaka u razredu PlayerController.

## Funkcija za stvaranje objekata scene

U kodu 5.4.1 prikazano je stvaranje kutije u sceni unutar funkcije generateCubicles. Kutija se stvori i spremi u UE-ov objekt tipa World, te se pokazivač na novostvoreni objekt koristi za postavljanje lokacije i dodavanje zapisa u rječnik igre. Kod 5.4.2 prikazuje stvaranje i transformaciju podloge te postavljanje parametara vezanih uz nju.

if (map.MAP\_MATRIX[y][x] == 'B') {

ACubicle\* BoxCubicle = GetWorld()->SpawnActor<ACubicleBox>(FVector(x \* floorTileX, y \* floorTileY, 121.0f), FRotator::ZeroRotator);

if (BoxCubicle) {

BoxCubicle->currentLocation = FVector2D(x, y);

tempString = FString::FromInt(x) + "//" + FString::FromInt(y);

GameMap.Add(tempString, true);

}

else return 1;

}

Kod 5.4.1 – stvaranje objekta kutije

AFloor\* FloorTile = GetWorld()->SpawnActor<AFloor>(FVector(0.0f, 0.0f, 0.0f), FRotator::ZeroRotator);

if (!FloorTile) return 1;

FVector FloorOrigin;

FVector FloorExtent;

FloorTile->GetActorBounds(false, FloorOrigin, FloorExtent);

float floorTileX = (FloorExtent.X \* 2.0f);

float floorTileY = (FloorExtent.Y \* 2.0f);

FloorTile->SetActorScale3D(FVector(width, height, 1.0));

FloorTile->AddActorLocalOffset(FVector(((floorTileX \* width) / 2) - FloorExtent.X, ((floorTileY \* height) / 2) - FloorExtent.Y, 0.0f));

mapCenter = FloorOrigin;

longestSide = FMath::Max(FloorExtent.X, FloorExtent.Y);

Kod 5.4.2 – postavljanje objekta podloge

## Funkcija za brisanje objekata scene

Funkcija DeleteMap poziva se prije svakog generiranja nove mape, jer se svi stvoreni objekti moraju maknuti iz svijeta prije nego se stvore novi. To se radi tako da se iterira kroz sve postojeće objekte i one koji su dio generirane razine se uništava, nakon čega se dereferencira pokazivač na odabrani objekt kako ne bi došlo do neovlaštenog pristupa memoriji, što je prikazano u kodu 5.5.

for (TActorIterator<APawn> PawnItr(GetWorld()); PawnItr; ++PawnItr) {

if (PawnItr->GetName().Left(7) == TEXT("Cubicle")) PawnItr->Destroy();

}

for (TActorIterator<AActor> ActorItr(GetWorld()); ActorItr; ++ActorItr) {

if (ActorItr->GetName().Left(5) == TEXT("Floor")) ActorItr->Destroy();

}

controller->dereferenceSelected();

Kod 5.5 – Funkcija za uništenje stvorenih objekata

# Obrada korisničkog unosa

Razred SparkyPlayerController zadužen je za postavljanje povratno-pozivnih funkcija i njihovo pozivanje prilikom obrade korisničkog unosa. Funkcije se postavljaju u konstruktoru tako što se pridruže komponenti zaduženoj za to.

Tipke korisničkog unosa, radnje koje okidaju i pridružene funkcije:

* Lijeva tipka miša, odabir objekta za pomak, SelectObject
* W – pomak prema gore, SlideUp
* A – pomak ulijevo, SlideLeft
* S – pomak prema dolje, SlideDown
* D – pomak udesno, SlideRight
* R – ponovo učitavanje trenutne mape, RefreshMap
* Ctrl + R – učitavanje nove mape, ReloadMap
* 1 – promjena perspektive, ChangeCamera
* Num 4 – smanjenje širine, SubtractWidth
* Num 6 – povećanje širine, AddWidth
* Num 2 – smanjenje visine, SubtractHeight
* Num 8 – povećanje visine, AddHeight
* Num 5 – povećanje broja linija ispravnog rješenja, AddLines
* Ctrl + Num5 – smanjenje broja linija ispravnog rješenja, SubtractLines

## Odabir objekta

Odabir objekta implementiran je tako da kada igrač lijevim klikom miša pritisne mjesto na ekranu oko središta željenog objekta, taj objekt postane „odabran“, što se istovremeno očituje zamjenom aktivnog materijala istaknutom inačicom običnog. Kako bi ta provjera bila moguća, koriste se Unrealove funkcije GetMousePosition, kojom nalazimo položaj pokazivača miša na ekranu, te ProjectWorldToScreen, kojom nalazimo koordinate objekata u svijetu projiciranih na ekran. Odabrani objekt je tada spreman za kretanje. Neposredno nakon odabira objekta, ako je prije toga neki drugi objekt bio odabran, on gubi taj status i materijal mu se vraća na staro. Kod 6.1 prikazuje cjelokupnu funkciju SelectObject.

FVector2D screenPosition;

FVector2D mousePosition;

int windowSizeX;

int windowSizeY;

GetMousePosition(mousePosition.X, mousePosition.Y);

GetViewportSize(windowSizeX, windowSizeY);

for (TActorIterator<ACubicle> PawnItr(GetWorld()); PawnItr; ++PawnItr) {

if (UGameplayStatics::ProjectWorldToScreen(this, PawnItr->GetActorLocation(), screenPosition)) {

if ((mousePosition - screenPosition).Size() < 25) {

if (SelectedCubicle) {

SelectedCubicle->bIsSelected = false;

SelectedCubicle->ToggleMaterial();

}

PawnItr->bIsSelected = true;

PawnItr->ToggleMaterial();

SelectedCubicle = \*PawnItr;

}

}

}

Kod 6.1 – Funkcija za odabir objekta u sceni

## Pomak objekta

Pomak objekta jedino je moguć ako je igra u stanju IN\_PROGRESS, a za vrijeme kretanja stanje se postavlja u IN\_MOTION. To je zato da ne bi bilo moguće započeti novi pokret prije nego se prethodni završi. Obrada kretanja sastoji se od izračuna udaljenosti pomaka do koje se dolazi iteracijom po poljima mape u smjeru zadanog pokreta dok se ne nađe prvo polje na kojem se nalazi drugi objekt, ili dok se ne dođe do ruba mape. Zatim se poziva metoda SlideMove razreda Cubicle te se ažurira rječnik igre i ostale nužne vrijednosti. Opisani postupak za pomak prema gore prikazan je u kodu 6.2.

if (gamemode->state != 0) return;

if (SelectedCubicle) {

FString tempString;

int currentX = SelectedCubicle->currentLocation.X;

int currentY = SelectedCubicle->currentLocation.Y;

gamemode->state = 3;

for (int x = currentX + 1; x < gamemode->GetWidth(); x++) {

tempString = FString::FromInt(x) + "//" + FString::FromInt(currentY);

if (gamemode->GameMap.Contains(tempString) && gamemode->GameMap[tempString] == true) {

if (x == currentX + 1) {

gamemode->state = 0;

return;

}

SelectedCubicle->SlideMove(FVector((x - 1) \* 300.0f, currentY \* 300.0f, 121.0f));

tempString = FString::FromInt(currentX) + "//" + FString::FromInt(currentY);

gamemode->GameMap[tempString] = false;

tempString = FString::FromInt(x - 1) + "//" + FString::FromInt(currentY);

if (gamemode->GameMap.Contains(tempString)) gamemode->GameMap[tempString] = true;

else gamemode->GameMap.Add(tempString, true);

SelectedCubicle->currentLocation.X = x - 1;

gamemode->state = 0;

return;

}

}

SelectedCubicle->SlideMove(FVector((gamemode->GetWidth() + 5) \* 300.0f, currentY \* 300.0f, 121.1f));

tempString = FString::FromInt(currentX) + "//" + FString::FromInt(currentY);

gamemode->GameMap[tempString] = false;

gamemode->state = 0;

}

Kod 6.2 – Pomak objekta prema gore

# Zaključak

Produkt rada je logička računalna igra napravljena većinom pišući c++ kod, koja se oslanja na dužu sjednicu igranja većeg broja razina kako bi ostvarila zadovoljavajući stupanj zahtjevnosti i igračevog zadovoljstva. Iako Unreal Engine nudi opsežan izbor alata za ostvarivanje standardnih obrazaca grafičkih aplikacija, korištenje istih izravno putem koda nije nužno uvijek intuitivno. Neke stvari poput generiranja objekata i postavljanja materijala su mi bile uvelike olakšane, no mogućnost pritiskanja mišem na objekt ili pomaka objekta sam ostvario na staromodni način jer korištenje posebnih funkcija koje bi u teoriji ponudilo bolji rezultat, u praksi nije radilo uopće.

Rad je u velikoj mjeri otvoren za buduća unapređenja, kao što video igre po prirodi jesu. Algoritam bi se mogao optimizirati tako da osigura zadani broj potrebnih pomaka da bi se uspješno prošla razina. Također, moglo bi se dodati još vrsta kutija (npr. nepomične, odbijajuće, odskočne…) ili čak drugih vrsta objekata. Osim toga, dodavanje mrežnih elemenata, kao što je čuvanje baze najboljih rezultata rješavanja razina s određenim sjemenom ili postojanje više igrača na istoj mapi.

# Literatura

1. Wikipedia, Puzzle Video Game. Poveznica: <https://en.wikipedia.org/wiki/Puzzle_video_game>
2. Wikipedia, Sokoban. Poveznica: <https://en.wikipedia.org/wiki/Sokoban>
3. Wikipedia, Unreal Engine. Poveznica: <https://en.wikipedia.org/wiki/Unreal_Engine>
4. Unreal Engine 4 Documentation. Poveznica: <https://docs.unrealengine.com/4.27/en-US/>
5. Unreal Engine Forum. Poveznica: <https://forums.unrealengine.com/categories?tag=unreal-engine>

# Sažetak

---

# Summary

---

# Skraćenice

---

# Privitak

---