Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Matej Terdič

Uporaba tehnologije XNA v razvoju iger za sistem Windows Phone

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: doc. dr. Mojca Ciglarič

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00491/2013
Datum: 09.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MATEJ TERDIČ**

Naslov: **UPORABA TEHNOLOGIJE XNA V RAZVOJU IGER ZA SISTEM WINDOWS PHONE**

USING XNA TECHNOLOGY IN DEVELOPMENT OF WINDOWS PHONE GAMES

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Preučite arhitekturo operacijskega sistema Windows Phone ter tehnologije in orodja, ki se največ uporabljajo za razvoj iger za ta sistem. Posebej se posvetite ogrodju XNA in opišite različne implementacije ogrodja. Raziščite možnosti uporabe odprtokodne implementacije ogrodja XNA za razvoj igre tipa "tower defense". V ta namen pojasnite žanr tovrstne igre, predstavitev sveta in principe iskanja poti. Igro tudi implementirajte, preizkusite in ovrednotite prednosti in slabosti uporabe izbrane tehnologije.

Mentor:
M. Ciglarič
doc. dr. Mojca Ciglarič

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Matej Terdič, z vpisno številko **63080187**, sem avtor diplomskega dela z naslovom:

Uporaba tehnologije XNA v razvoju iger za sistem Windows Phone

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Mojce Ciglarič,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 12. septembra 2013 Podpis avtorja:

Zahvaljujem se mentorici doc. dr. Mojci Ciglarič za njeno pomoč, teoretično podlago in nasvete pri izdelavi diplomske naloge.

Zahvaljujem se vsem profesorjem in asistentom, ki so me učili v teku mojega študija, za pridobljeno znanje in navdih za nadaljnji študij.

Posebna zahvala gre tudi moji družini in prijateljem, ki so me podpirali in spodbujali v času študija.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Tower defense **3**

2.1 Igralnost 3

2.2 Zgodovina 4

2.3 Znane igre 6

2.4 Minedefense 11

3 Razvojna orodja **15**

3.1 C# in .NET 15

3.2 Visual Studio 16

3.3 Paint.NET 17

3.4 Windows Phone SDK 17

3.5 Windows Phone 18

4 Tehnologija XNA **23**

4.1 Opis tehnologije XNA 23

4.2 Knjižnica XNA 25

4.3 Knjižnica XNI 27

4.4 Knjižnica MonoGame 28

5 Umetna inteligenca iger tower defense **31**

5.1 Predstavitev preiskovalnega prostora 31

5.2 Algoritmi za preiskovanje sveta 33

6 Uporaba XNA v razvoju igre Minedefense **39**

6.1 Osnovni razredi in metode knjižnice XNA 39

6.2 Glavna zanka igre 42

6.3 Scene in stanja igre 44

6.4 Predstavitev entitet 46

6.5 Ozemlje in iskanje poti 51

6.6 Sovražnik 52

6.7 Igralec 56

6.8 Uporabniški vmesnik 59

7 Sklepne ugotovitve **61**

7.1 Razvoj iger brez XNA 62

7.2 Izboljšave igre Minedefense 62

Povzetek

V diplomski nalogi je prikazana uporaba tehnologije XNA v razvoju igre za operacijski sistem Windows Phone 8. Za lažje razumevanje igre je na začetku opisan žanr tower defense, kateremu pripada tudi razvita igra Minedefense. Prikazan je princip igranja takšne igre, zgodovina žanra ter primerjava igre Minedefense z ostalimi igrami tower defense. Nato sledi opis tehnologij in orodij, ki smo jih uporabili pri izdelavi igre. V tem delu je na kratko prikazana arhitektura operacijskega sistema Windows Phone. Temu sledi prikaz delovanja ogrodja XNA in opis različnih implementacij ogrodja. Pred opisom dejanskega razvoja igre so opisani tudi različni načini predstavitve sveta in principi iskanja poti v video igrah tipa tower defense. Na koncu je prikazan potek razvoja igre z uporabo odprtokodne implementacije ogrodja XNA.

Ključne besede: razvoj mobilne igre, tehnologija XNA, knjižnica MonoGame, Windows Phone, tower defense, iskanje poti

Abstract

The thesis illustrates the use of XNA technology to develop video game for Windows Phone 8 operating system. To facilitate understanding of the game the tower defense genre is described first. This section describes the principles of gameplay, history of the genre and compares the game Minedefense with some other games of the same genre. Then follows the description of the technologies and tools we used during the development. This section briefly shows the architecture of the Windows Phone operating system. This is followed by the demonstration of the XNA framework and the description of implementations of this framework. Before we get to describe an actual process of the development, different ways to present the world and the pathfinding algorithms are explained. The last part of the thesis describes the process of the development using open-source implementation of XNA framework.

Keywords: mobile game development, XNA technology, MonoGame framework, Windows Phone, tower defense, pathfinding

Poglavje 1

Uvod

Mobilne naprave so se v zadnjih letih zelo razširile med uporabniki in napredovale v zmogljivosti. Slednja se še vedno hitro povečuje in današnje mobilne naprave lahko že skoraj enačimo z osebnimi računalniki. Ravno priljubljenost med uporabniki in zmogljivost sta največ pripomogli k kvaliteti in številu video iger. Mnogo znanih naslovov, nekoč prisotnih le na osebnih računalnikih in igralnih konzolah, danes najdemo tudi na mobilnih napravah.

Trenutno najnovejši priljubljen mobilnih operacijskih sistem je Microsoft Windows Phone. Aplikacija Xbox Marketplace razvijalcem video iger omogoča objavo izdelkov, uporabnikom sistema Windows Phone pa enostaven nakup in namestitev iger. Ker je Windows Phone nov sistem, se zenkrat zanj zanima relativno malo razvijalcev. Posledično trg še ni prenasičen z aplikacijami in igrami. Nenasičenost, in predvsem zanimiv pristop novega operacijskega sistema k mobilni tehnologiji, je ob mojem začetku razvoja video iger botrovalo izbiri ciljne platforme. Tretji razlog je bila popularnost knjižnice XNA za razvoj enostavnih iger za platforme Xbox, Windows in Windows Phone. Z uporabo te knjižnice so neodvisni razvijalci iger samostojno razvijali preproste igre brez poznavanja kompleksnejših tehnologij, kot sta DirectX in OpenGL. S prihodom operacijskih sistemov Windows 8 in Windows Phone 8 pa je podjetje Microsoft najavilo konec razvoja te priljubljene knjižnice. Razvijalci iger smo bili zato primorani poiskati alternative.

Ena izmed alternativ je uporaba knjižnice MonoGame. Gre za odprtokodno knjižnico, ki je skoraj identična knjižnici XNA. V ozadju uporablja enako tehnologijo (OpenGL namesto DirectX in Mono namesto .NET) in za razvoj iger lahko uporabljamo isti programski jezik (C#). Teoretično je programska koda igre XNA povsem kompatibilna s programsko kodo igre MonoGame. Uporabo knjižnice MonoGame, kot implementacijo tehnologije XNA, bomo prikazali skozi razvoj igre Minedefense za Windows Phone 8. Minedefense je strateška igra, ki žanru tower defense prinaša nekatere novosti. Te novosti bolj ali manj spremenijo igralnost tipične igre tower defense in naredijo igro Minedefense unikatno.

Tehnologijo XNA, operacijski sistem Windows Phone, lastnosti igre Minedefense in primerjavo z ostalimi igrami žanra si bomo podrobneje ogledali v naslednjih poglavjih.

Poglavje 2

Tower defense

Poglavje opisuje splošne lastnosti igre tower defense, zgodovino žanra in prikazuje primerjavo razvite igre Minedefense s priznanimi igrami tower defense.

2.1 Igralnost

Glavna prepoznavnost igre tower defense [1] sta gradnja obrambnih stolpov in sovražne enote, ki prodirajo proti igralčevi bazi. Sovražnikove enote prihajajo v valovih in vsaka enota, ki uspešno prodre do baze, igralcu vzame življenje. Če igralec izgubi vsa življenja, se igra zaključi s porazom. Sredstva za gradnjo novih ali nadgradnjo obstoječih stolpov igralec pridobi z ubijanjem sovražnikovih enot, v nekaterih izvedenkah pa tudi ob koncu vsakega sovražnikovega napada. Ko igralec odbije vse sovražnikove napade, je igra dobljena.

Pot, po kateri se giblje sovražnikova enota, je lahko vnaprej začrtana, lahko pa se sproti izračunava. Pri prvem načinu je pot striktno ločena od ostalega dela terena, ki je namenjen gradnji stolpov. Strateška postavitev stolpov je v tem primeru pomembna zaradi razporeditve napada. Igralec želi stolpe locirati tako, da bo vsak stolp pokrival čimvečji del poti - večji kot bo domet stolpa, več škode bo lahko prizadejal sovražniku. Poleg načina ločene poti obstaja tudi način, ko je igralec primoran graditi stolpe na sami

poti. V tem primeru lahko ob pravilni postavitvi stolpov nastane labirint, ki ga morajo sovražnikove enote prehoditi. Dobro zasnovan labirint lahko sovražniku podaljša pot in s tem omogoči stolpom več časa za napadanje.

Tower defense se lahko igra v eno ali več igralskem načinu. V večigralskem načinu obstajata dve glavni veji iger: igre, kjer igralci branijo skupno bazo in igre, kjer vsak igralec brani svojo bazo. Ko igralci branijo skupno bazo, morajo med seboj uskladiti postavitev stolpov, da sovražne enote čim bolj enakomerno dosežejo vse igralce. V nasprotnem primeru lahko nekateri igralci ostanejo brez sredstev za gradnjo in posledično kasneje v igri ne morejo več zaustaviti sovražnikovih enot. Igra se zaključi, ko igralcem zmanjka skupnih življenj, ali ko odbijejo vse napade. Ko posamezen igralec brani svojo bazo, je lahko igra zasnovana tako, da sovražnikove enote pošilja računalnik, lahko pa tudi tako, da si enote med seboj pošiljajo igralci sami. V slednjem primeru mora igralec pravilno razporediti sredstva med gradnjo stolpov in nakup enot, ki jih pošlje soigralcu. Tukaj se igra zaključi takrat, ko vsi igralci - razen zmagovalca - ostanejo brez življenj.

2.2 Zgodovina

Prva igra, ki jo lahko imenujemo tower defense, je igra Rampart [2], razvita leta 1990 v podjetju Atari Games. Od današnjih tower defense iger se razlikuje po tem, da je moral igralec sam upravljati s stolpi in meriti v sovražnike.

Na začetku igre je igralec izbral območje na katerem je nato nastal grad, vidno na sliki 2.1. Znotraj gradu je lahko gradil stolpe. Gradnji stolpov je sledil sovražnikov napad. To so bile ladje, ki so obstreljevale grad, igralec pa jih je z ročnim upravljanjem stolpov uničeval. Določene ladje so ob pristanku na kopno spustile dodatne sovražnike - pehoto. Ko je igralec uničil vse ladje, in morebitne dodatne enote, je sledila časovno omejena faza popravljanja gradu. Igralec je dobil na voljo gradnike različnih oblik, ki jih je moral pravilno umestiti v poškodovane dele gradu. Če mu je uspelo popraviti

Slika 2.1: Igra Rampart [2].

celoten grad, je lahko postavil nove stolpe in nadaljeval z igro. V nasprotnem primeru je bil poražen in igra se je zaključila.

Začetki žanra tower defensa, kot ga poznamo danes [3], segajo v leto 2001. Tega leta je pri razvijalcu in založniku Blizzard Entertainment izšla strateška igra Warcraft III. Čeprav ni vsebovala nikakršnih sledi tower defensa je prav ta igra najbolj vplivala na razvoj žanra. Bila je zasnovana tako, da so lahko igralci s pomočjo aplikacije World Editor sami razvijali in spreminjali zemljevide. Tako so nekateri uporabniki začeli razvijati lastne zemljevide, med njimi tudi takšne za igro tower defense. Ti zemljevidi so hitro postali priljubljeni in kmalu so se na spletu pojavile enostavnejše različice iger tower defense, narejene za vmesnik Flash. Danes igre tower defense najdemo že na praktično vseh platformah: osebnih računalnikih, pametnih telefonih, tablicah, igralnih konzolah in na spletu.

Slika 2.2: Igra Warcraft III z zemljevidom Element Tower Defense.

2.3 Znane igre

Element Tower Defense

Element Tower Defense [4] je najbolj razširjen in igran tower defense zemljevid za igri Warcraft III (slika 2.2) in StarCraft II. Omogoča istočasno igranje največ osmih igralcev. Vsak igralec brani svojo bazo.

Glavna lastnost igre je 6 elementov: svetloba, tema, voda, ogenj, narava ter zemlja. Vsaka enota in vsak stolp pripadata vsaj enemu tipu elementa. Vsak element ima svoj nasprotni element, kar pomeni, da stolp določenega tipa hitreje uničuje enote nasprotnega tipa (npr. voda proti ognju), ali pa jim povzroča samo polovično škodo (npr. ogenj proti vodi). Po vsakem petem napadu igralec pridobi nov tip elementa, ki ga lahko uporabi za razvoj stolpov novega tipa. Več različnih elementov (največ 3) lahko tudi združi in tako pridobi močnejše stolpe. Takšni stolpi imajo posebne moči, kot so upočasnevanje sovražnikov, napadanje večih sovražnikov istočasno in podobno.

Igra se zaključi, ko igralci - razen enega - izgubijo svoja življenja oziroma ko igralci ubranijo vse sovražnikove napade. V tem primeru so zmagovalci vsi, ki so preživeli vse napade.

Wintermaul TD

Wintermaul TD [5] je priljubljen zemljevid za igri Warcraft III in StarCraft II. Namenjen je največ devetim igralcem, ki branijo skupno bazo.

V nasprotju z večino tower defense iger je pot, po kateri hodijo sovražnikove enote, namenjena tudi gradnji stolpov. Z dobro strateško postavitvijo stolpov v obliki labirinta se lahko sovražnikova pot precej podaljša. Posledično imajo stolpi več časa za napadanje. Pri tem mora igralec paziti, da sovražniku še vedno pusti dovolj prostora, da ima le-ta prosto pot do baze. V nasprotnem primeru bo sovražnik začel uničevati stolpe, in bo pot nadaljeval šele, ko bodo vsi stolpi uničeni. Takšni situaciji skoraj zagotovo sledi poraz, kajti brez stolpov bodo sovražnikove enote brez oviranja prispele do baze. Igra vsebuje tudi leteče enote, ki letijo nad stolpi in se tako izognejo sprehodu skozi labirint.

Igralci branijo skupno bazo in imajo skupna življenja. Igra se konča, ko jim zmanjka življenj, ali ko obranijo bazo pred vsemi sovražnikovimi napadi.

Legion TD

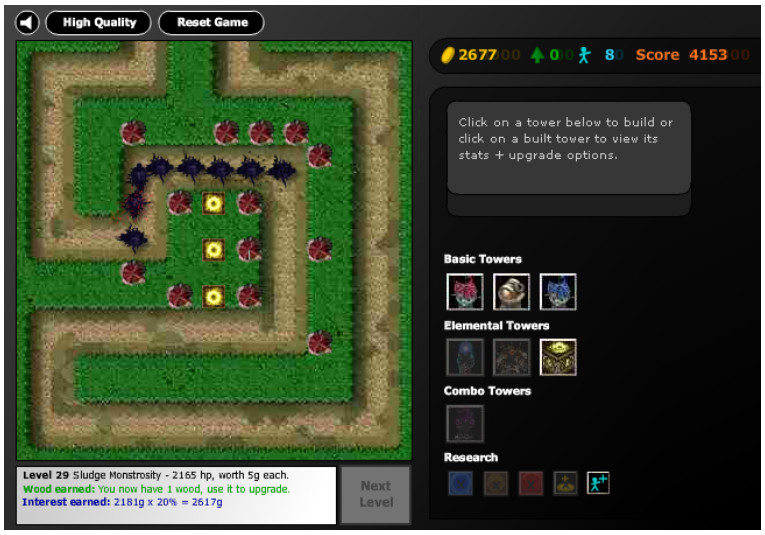
Tudi Legion TD [6] je zemljevid za igri Warcraft III in StarCraft II. Namenjen je največ osmim igralcem, ki se razdelijo v 2 skupini

Igra združuje lastnosti predhodno opisanih Element Tower Defense in Wintermaul TD. Podobno kot v igri Wintermaul TD lahko igralci z gradnjo labirintov ovirajo sovražnika. Vendar pa igralci ne branijo skupne baze, ampak, podobno kot pri igri Element Tower Defense, tekmujejo med seboj. Razdeljeni so v 2 ekipi, znotraj katerih igralci med seboj sodelujejo.

Zmaga tista ekipa, ki ji ne zmanjka življenj. Če obe ekipi ubranita vse napade tudi obe ekipi zmagata.

Flash Element Tower Defense

Flash Element Tower Defense [7] je najbolj igrana tower defense igra.



Slika 2.3: Spletna igra Flash Element Tower Defense.

Gre za preprosto igro, narejeno v programskem okolju Flash (slika 2.3), kar omogoča igranje v spletnem brskalniku. Namenjena je enemu igralcu.

Izdelana je po vzoru predhodno opisanege igre Element Tower Defense, vendar ni tako kompleksna. Vsebuje manjše število stolpov in sovražnikov. Posledično je enostavnejša za razumevanje in ni potrebno veliko časa, da igralec osvoji način igranja. Kot pri igri Element Tower Defense vsaka enota in vsak stolp pripadata enemu tipu elementa in vsak element ima svoj nasprotni element. Z združevanjem elementov lahko igralec sestavi močnejše stolpe.

Igra se zaključi, ko igralec izgubi vsa življenja, ali ko ubrani vse sovražnikove napade. V obeh primerih se lahko vpiše v spletno lestvico in se na ta način primerja z ostalimi igralci.

Desktop Tower Defense

Desktop Tower Defense [8] je, tako kot predhodno opisana igra Flash Element Tower Defense, izdelana v programskem okolju Flash in je namenjena enemu igralcu.

Igra je drugačna od predhodno opisanih tower defense iger. Sovražnikova pot ni striktno določena, ampak je celotna igralna površina na začetku prazen pravokotnik. S postavitvijo stolpov igralec začrta pot, ki jo mora sovražnik prehoditi. Točka, ki jo mora sovražnik doseči, se nahaja na nasprotni strani pravokotnika kot točka sovražnikova vstopa na igralno površino. Obstaja več vrst sovražnikov: nekateri so odporni na določene stolpe, nekateri lahko letijo in se ne gibljejo po labirintu, spet drugi se gibljejo hitreje in podobno.

Igra se zaključi, ko igralec izgubi vsa življenja, ali ko ubrani vse napade.

Defense Grid: The Awakening

Zanimiva igra je tudi Defense Grid: The Awakening [9]. Namenjena je igranju na osebnem računalniku ali igralni konzoli Xbox 360. Igra jo en igralec.

Največja posebnost igre je dvosmerna pot sovražnika. To pomeni, da igralec ne izgubi življenja, ko sovražnik prodre do baze, ampak šele, ko slednji ponovno prispe na začetek poti. Sovražnik iz baze ukrade energijsko jedro in če mu to jedro uspe prenesti do začetka poti, igralec jedro izgubi. Eno jedro je v tem primeru enakovredno enemu življenju. Stolpi sovražnike napadajo na poti proti bazi in na poti proti začetku. Če sovražnika ubijejo ko le-ta prenaša jedro, se jedro počasi začne vračati proti bazi. Naslednjemu sovražniku ni potrebno priti do baze, ampak lahko pobere jedro, ki ga je njegov predhodnik izgubil na poti. Igralec ima na voljo tudi močan laserski napad, s katerim uniči vse prisotne sovražnike. Po napadu laser potrebuje veliko časa, da ponovno postane na voljo.

V opisani igri igralčeva življenja predstavljajo energijska jedra, ki jih sovražnik poskuša ukrasti. Ko igralec izgubi vsa jedra, se igra konča. Če ima po koncu zadnjega napada v bazi vsaj še 1 jedro, je igra dobljena.

Plants vs. Zombies

Plants vs. Zombies [10] je priljubljena in enostavna tower defense igra.

Slika 2.4: Igra Plants vs. Zombies.

Lahko jo igra en sam igralec, omogoča pa tudi večigralstvo. Na voljo je na vseh bolj znanih platformah.

Za igro Plants vs. Zombies je specifično, da sovražnikove enote - zombiji - prihajajo v ravni liniji iz desne proti levi, kar lahko opazimo na sliki 2.4. Če sovražnik prodre do skrajno leve pozicije vzame igralcu eno življenje. Igralec v ravnih linijah gradi stolpe - rastline - ki branijo levo stran sveta. Stolpi imajo različne vloge: nekateri napadajo zombije, drugi jih upočasnijo, spet tretji pa so utrjeni proti udarcem in nimajo sposobnosti napada. Ko zombi prispe do stolpa, ga začne napadati. Če mu uspe uničiti stolp, nadaljuje pot proti levemu delu sveta, oziroma naslednjemu stolpu v liniji.

Ko igralcu zmanjka življenj, ali ko se ubrani vseh napadov, se igra zaključi.

Orcs Must Die!

Nekoliko drugačna je igra Orcs Must Die! [11]. Omogoča eno igralski način, na voljo je za osebne računalnike in igralno konzolo Xbox 360.

Obstaja tudi nadaljevanje Orcs Must Die! 2, ki omogoča igranje dvema igralcema.

Zanimiv je način pogleda na igro. Za običajen tower defense je značilen pogled iz ptičje perspektive, pogled v igri Orcs Must Die! pa je prvoosebni. Tudi gradnja stolpov je nekoliko drugačna. Igralec nadzoruje lik, ki ima sposobnost graditi različne pasti, na katere lahko namesti tudi orožje. Te pasti, ki sovražniku otežujejo napad na središče baze, predstavljajo stolpe. Pasti so različne: lahko so sulice, ki se dvigajo iz tal, lahko so buzdovani, ki se vrtijo in podobno.

Kot pri ostalih opisanih igrah lahko igralec zmaga tako, da se ubrani vseh napadov. Če mu pred tem zmanjka življenj, izgubi.

2.4 Minedefense

Igra Minedefense, na sliki 2.5, vsebuje večino lastnosti klasične tower defense igre. Večja in manjša odstopanja pa jo naredijo unikatno. Med vidnejše razlike štejemo večje število različnih sredstev za gradnjo ter vzdržjivost stolpov.

Osnovna tematika izvira iz igre Minecraft [12]. Minecraft je popularna puščavniška igra, razvita leta 2011 kot samostojni projekt programerja Markussa Persona. Omogoča igranje v eno in več igralskem načinu. Svet v igri se generira naključno in je sestavljen iz raznovrstnih kock (slika 2.6). Vsaka kocka predstavlja določen material ali objekt in ima unikaten učinek na igralca. Na prvi pogled izgleda Minecraft zelo preprosta igra: igralec lahko s posebnimi orodji vzame kocko in jo prestavi na drugo lokacijo, ali pa iz nje sestavi nova orodja in materiale. Njeno kompleksnost opazimo šele, ko se poglobimo v igro. Različnih kock v igri je preko 100. Če poleg štejemo še orodja, hrano in podobno ta številka še precej naraste. Vsaka nova kocka in vsako orodje se sestavi po svojevrstnem receptu in uporabi pri sestavi drugih kock in orodij. Med pomembnejše kocke štejemo les, kamen, zemljo, pesek, premog, železo, zlato in diamant. Našteti materiali, razen zemlje in peska, se uporabijo pri sestavljanju različnih orodij. Med pomembnejša orodja šte-

Slika 2.5: Razvita igra Minedefense.

Slika 2.6: Igra Minecraft.

jemo kramp, lopato in sekiro. Z izbiro pravega orodja lahko igralec iz okolja hitreje pridobi kocke. Če potrebuje les, je najboljše orodje sekira. Za kamen in ostale rudnine uporabi kramp. Zemljo in pesek najhitreje pridobi z uporabo lopate. Hitrost pridobivanja kock in vzdržljivost orodja sta odvisni tudi od materiala, iz katerega je orodje sestavljeno. Diamantno orodje je daleč najbolj vzdržljivo in z njim igralec hitro pridobi kocko. Leseno orodje se bo prej uničilo in več časa je potrebnega, da igralec pridobi kocko.

Opisane lastnosti igre Minecraft so značilne tudi za igro Minedefense:

- Igra vsebuje **3 tipe stolpov**: kramp, lopato in sekiro. Vsak tip stolpa lahko napada samo določene enote. Peščene enote in enote, sestavljene iz zemlje, bodo napadene samo s strani stolpov tipa lopata. Lesene enote bodo napadali samo stolpi tipa sekira. Vse ostale enote pa bodo napadali stolpi tipa kramp.

- Za gradnjo stolpov se uporablajo **različni materiali**. Pri klasičnih tower defense igrah igralec razpolaga z enim sredstvom (tipično denarjem) za gradnjo stolpov. Pri igri Minedefense pa igralec uporablja 5 različnih sredstev: les, kamen, železo, zlato in diamant.

- **Število enot materiala**, potrebnega za izgradnjo stolpa, je odvisno

od tipa stolpa. Za katerikoli stolp igralec potrebuje najmanj 2 enoti lesa. Poleg lesa potrebuje za stolp tipa lopata še 1 enoto poljubnega materiala, za stolp tipa kramp ali sekira pa 3 enote poljubnega materiala.

- **Moč in vzdržljivost** stolpa sta odvisni od materiala, uporabljenega pri gradnji stolpa. Za klasične tower defense igre je značilno, da so zgrajeni stolpi večni. Pri igri Minedefense pa stolpu z vsakim napadom pade vzdržljivost. Ko vzdržljivost pade na 0, se stolp poruši. Najbolj vzdržljivi so diamantni stolpi. Sledijo železni, kamniti, leseni in zlati. Zlati stolpi so najmočnejši. Sledijo diamantni, železni, kamniti in leseni.

- Vsaka **sovražnikova enota predstavlja določen material**. Če je enota lesena, kamnita, železna, zlata ali diamantna, bo igralec z njenim ubojem pridobil 1 enoto tega materiala.

- Sovražnikove enote so z vsakim novim valom **drugačne hitrosti, drugačne vzdržljivosti in drugačnega materiala**. Vse lastnosti enot so med seboj usklajene tako, da je vsak naslednji val nekoliko zahtevnejši od prejšnjega.

Poglavje 3

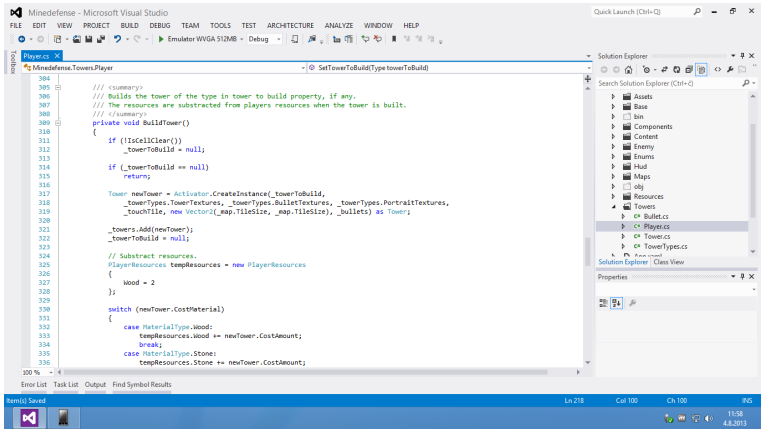
Razvojna orodja

Na začetku poglavja je opisan programski jezik C#, ki je primarni jezik za razvoj iger XNA in v katerem je bila razvita tudi igra Minedefense. Sledi opis razvojnih orodij Visual Studio, Paint.NET in Windows Phone SDK. Na koncu poglavja je opisan operacijski sistem Windows Phone.

3.1 C# in .NET

C# [13] je objektno usmerjen programski jezik s sintakso, podobno jezikom Java in C++. Je strukturiran, imperativen, objektno usmerjen, dogodkovno voden, statično in dinamično tipiziran, močno tipiziran in varno tipiziran programski jezik. C# je najbolj razširjen programski jezik za razvoj aplikacij na operacijskih sistemih Windows, Windows Server in Windows Phone. Njegov razvoj vodi podjetje Microsoft. Prva različica je bila razvijalcem na voljo leta 2000 kot del ogrodja .NET.

Programska koda, napisana v .NET, se ne prevede v binarno kodo fizičnega sistema, ampak v binarno kodo virtualnega sistema, imenovanega Common Language Runtime (CLR) [14]. Prevedeno kodo nato CLR med izvajanjem interpretira v binarno kodo fizičnega sistema. Na ta način se razvijalcem onemogoči neposreden dostop do fizičnega sistema, kar zmanjša število napak, prisotnih pri dostopu do fizičnih komponent sistema.

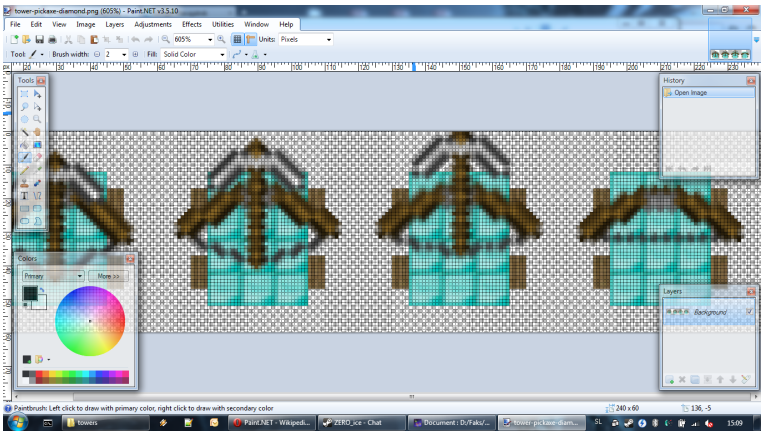


Slika 3.1: Microsoft Visual Studio 2012.

3.2 Visual Studio

Visual Studio [15] je primarno orodje za razvoj aplikacij za operacijske sisteme Windows, Windows Server in Windows Phone. Omogoča uporabo programskih jezikov C#, C++, Visual Basic in jezikov za spletnih razvoj. Orodje razvijajo pri podjetju Microsoft. Na voljo je v brezplačni ali plačljivi različici. Za razvoj aplikacij in iger za Windows Phone 8 potrebujemo različico Visual Studio 2012 (na sliki 3.1), ki je trenutno najnovejša različica na trgu.

Kot večina razvojnih orodij tudi Visual Studio vsebuje pripomočke za razvoj aplikacij. Urejevalnik kode omogoča paudarjanje sintakse, samodejno vstavljanje kode, zlaganje kode in refaktoriranje. Z orodjem za oblikovanje lahko razvijalec določi postavitev komponent v aplikaciji, izdeluje podatkovne sheme in celo gradi razrede. Razhroščevalnik omogoča postavljanje ustavitvenih točk, pregled trenutnih vrednosti v objektih in spremenljivkah, ročno izvajanje funkcij in pregled sklada vseh klicev do trenutne funkcije. Vsebuje tudi orodja za uporabo sistema SVN in ustvarjanje ter urejanje novih projektov.



Slika 3.2: Paint.NET 3.5.

3.3 Paint.NET

Paint.NET [16], na sliki 3.2, je brezplačen program za urejanje grafike. Rick Brewster je z razvojem aplikacije začel kot projekt v sklopu študija na univerzi v Washingtonu. Prva različica programa je izšla leta 2004.

Urejevalnik je enostaven za uporabo, saj ne vsebuje zahtevnih funkcij, ki jih imajo nekateri dražji urejevalniki. Kljub temu omogoča nekatere osnovne funkcije, kot so ostritev, zabrisovanje, izkrivljanje in podobno. Omogoča tudi nastavljanje svetlosti, kontrasta, odtenkov ter nasičenost. Kljub preprostosti urejevalnika lahko uporabnik grafiko ureja v različnih plasteh.

3.4 Windows Phone SDK

Windows Phone SDK vsebuje knjižnice za razvoj aplikacij za operacijski sistem Windows Phone. Tako kot Visual Studio in C# tudi Windows Phone SDK razvija podjetje Microsoft. Windows Phone SDK že vsebuje brezplačno različico orodja Visual Studio. Če orodja Visual Studio še nimamo nameščenega, se ob namestitvi Windows Phone SDK samodejno namesti brezplačna različica. V nasprotnem primeru se Windows Phone SDK integrira z obstoječo namestitvijo orodja Visual Studio. Za razvoj aplikacij za operacijski

sistem Windows Phone 8 potrebujemo Windows Phone 8 SDK.

Emulator operacijskega sistema Windows Phone 8 je vključen v Windows Phone 8 SDK. Omogoča nam testiranje mobilne aplikacije na osebnem računalniku. To pomeni, da ne potrebujemo fizične mobilne naprave za testiranje v fazi razvoja. Vseeno je priporočljivo testiranje na fizični napravi pred izdajo končne različice aplikacije.

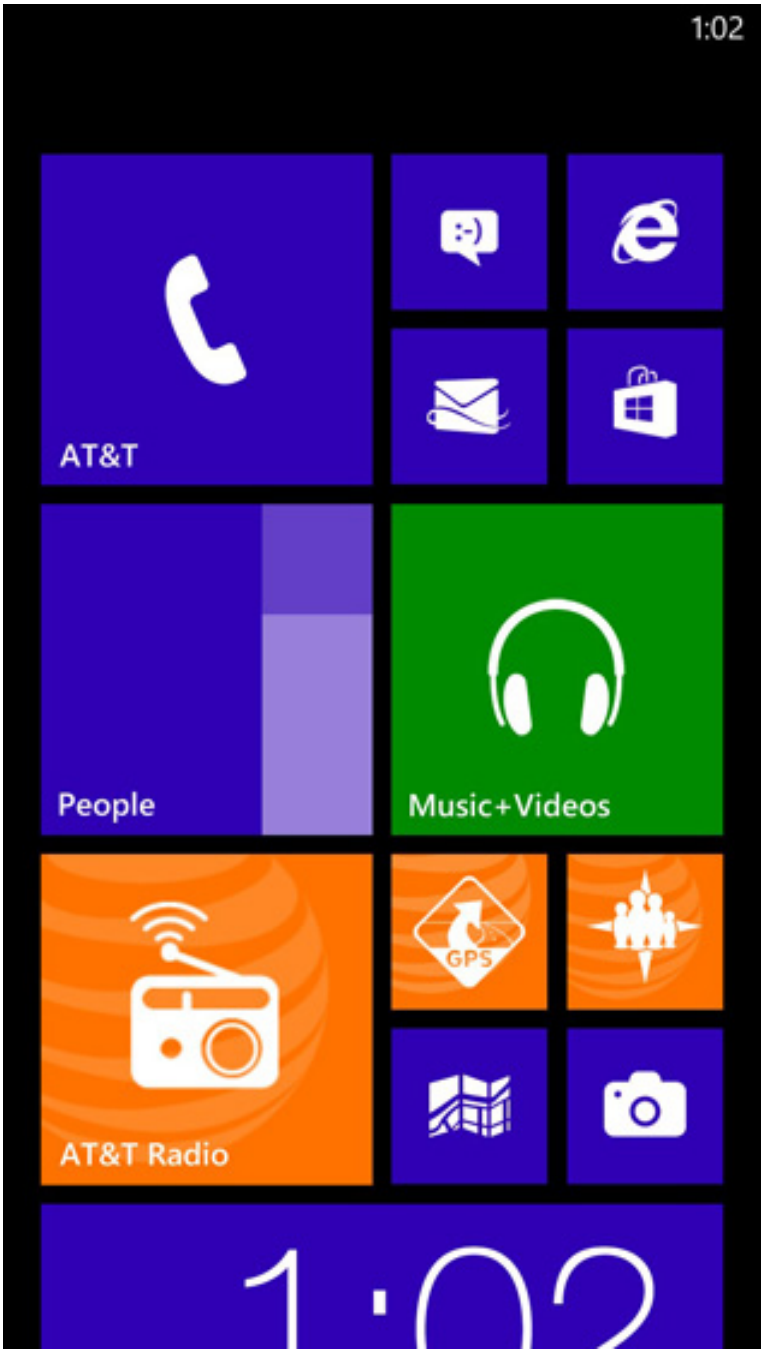
3.5 Windows Phone

Operacijski sistem Windows Phone [17] proizvajalca Microsoft je trenutno 3. najbolj razširjen mobilni operacijski sistem. Nameščen je na naprave različnih proizvajalcev strojne opreme, najpomembnejša partnerja pa sta Nokia in HTC. Prvi operacijski sistem iz družine Windows Phone, Windows Phone 7, je izšel 8. novembra 2010. Trenutno aktualna različica je Windows Phone 8, ki je izšla 29. oktobra 2012. Ker je naša igra izdelena za različico 8, se bomo v tem poglavju osredotočili na operacijski sistem Windows Phone 8.

3.5.1 Uporabniški vmesnik

Grafični uporabniški vmesnik operacijskega sistema Windows Phone [18] (na sliki 3.3 operacijski sistem Windows Phone 8) po izgledu in uporabi odstopa od uporabniških vmesnikov ostalih operacijskih sistemov. Med največje posebnosti vmesnika sodijo žive ploščice (ang. live tiles) in aplikacijski razdelki (ang. hubs).

Žive ploščice nadomestijo klasične ikone in pripomočke, kot jih poznamo pri ostalih operacijskih sistemih. Vsaka ploščica predstavlja povezavo do aplikacije. Le-to lahko uporabnik s klikom na pripadajočo ploščico tudi zažene. Vsaka ploščica prikazuje tudi določene informacije pripadajoče aplikacije. Kot primer vzemimo aplikacijo Koledar (ang. Calendar). S klikom na pripadajočo ploščico bo uporabnik zagnal aplikacijo Koledar z vsemi vnešenimi podatki. Nekatere osnovne podatke, vnešene za tekoči teden, pa lahko uporabnik hitro razbere iz ploščice brez zagona aplikacije. Velikost in po-



Slika 3.3: Začetni zaslon sistema Windows Phone 8.

stavitev ploščic lahko uporabnik prilagodi svojim potrebam - od velikosti je namreč odvisna količina prikazanih informacij. Ploščice najmanjše velikosti se večinoma uporablajo samo za zagon aplikacije. Večje ploščice pa lahko prikazujejo že zajetno količino informacij.

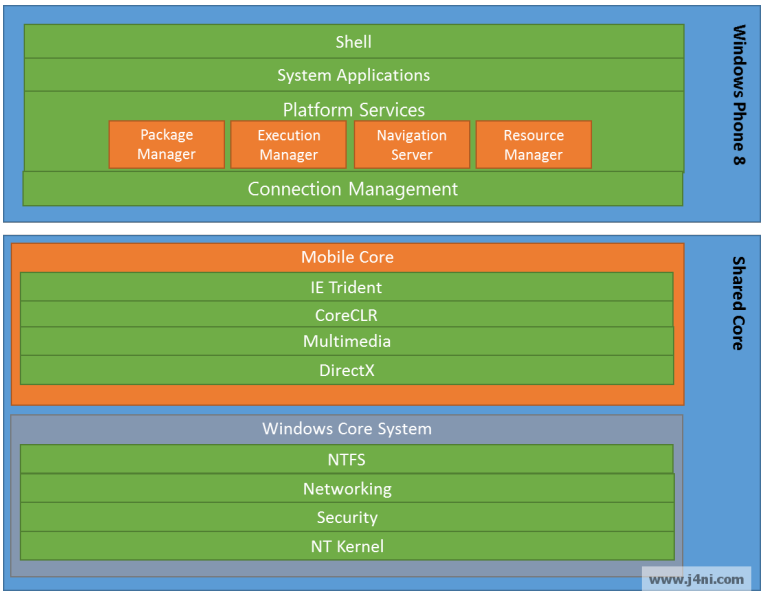
Druga omenjena posebnost uporabniškega vmesnika so razdelki. Pri ostalih operacijskih sistemih so aplikacije med seboj večinoma neodvisne. Razdelki v operacijskem sistemu Windows Phone pa med seboj združujejo aplikacije s podobno tematiko. Razdelek Ljudje (ang. People hub) lahko poleg kontaktov, shranjenih v telefonu, prikazuje tudi kontakte iz različnih socialnih omrežij. Vse na enem mestu. Uporabnik lahko z uporabo enega razdelka upravlja z več aplikacijami istočasno.

Uporabniški vmesnik operacijskega sistema Windows Phone je bil zasnovan predvsem za naprave, ki uporabljajo zaslon na dotik. Zgoraj opisane žive ploščice in aplikacijski razdelki zaradi svoje narave še dodatno prispevajo k dobri uporabniški izkušnji. Preklapljanje med različnimi deli razdelka je z uporabo prstov hitro in enostavno.

3.5.2 Zgradba operacijskega sistema

Windows Phone 8 se od svojih predhodnikov razlikuje v zgradbi operacijskega sistema [19]. Starejši sistemi Windows Phone bazirajo na jedru CE, Windows Phone 8 pa bazira na jedru NT. Jedro NT [20] je osnova tudi za operacijska sistema Windows in Windows Server. Glavne lastnosti operacijskega sistema Windows Phone z jedrom NT:

- večopravilnost,
- podpora večjedrnim procesorjem,
- podpora ločljivostim "WVGA 800x480 15:9", "WXGA 1280x768 15:9"in "720p 1280x720 16:9",
- podpora MicroSD spominskim karticam.



Slika 3.4: Zgradba operacijskega sistema Windows Phone 8 [19].

Operacijski sistem Windows Phone 8 je sestavljen iz 3 večjih gradnikov, kar je razvidno iz slike 3.4. Ti gradniki so jedro NT, mobilno jedro in dejanski operacijski sistem Windows Phone 8.

Jedro NT

Jedro NT je najosnovnejše jedro operacijskega sistema Windows Phone 8. Kot je navedeno na začetku podpoglavja, to isto jedro uporabljata tudi operacijska sistema Windows in Windows Server.

Jedro NT je sestavljeno iz sklopa rutin za dostop do strojne opreme, gonilnikov, ki preko rutin dostopajo do strojne opreme, ter storitev, ki uporabljajo gonilnike. Med te storitve spadajo storitev za nadzor nad vhodno/izhodnimi napravami, storitev za nadzor na pomnilnikov, storitev za nadzor nad procesi in podobno.

Glavna naloga jedra NT je torej nadzor nad strojno opremo. Vsaka operacija aplikacije lahko do strojne opreme dostopa samo preko jedra NT.

Mobilno jedro

Mobilno jedro je tisti del sistema Windows Phone, ki ne sovpada več popolnoma z ekvivalentnim delom operacijskih sistemov Windows in Windows Server. Mobilno jedro je okrnjena različica jedra Windows in Windows Server.

Mobilno jedro je sestavljeno iz programskih knjižnic DirectX, programskih knjižnic za multimedijo, sistema Trident za izvajanje XAML kode in prevajalnika CLR. Igra Minedefense se v tem delu operacijskega sistema, natančneje v prevajalniku CLR, po delih prevede (z uporabo JIT prevajalnika) iz strojne kode virtualnega sistema CLR v strojno kodo fizičnega sistema.

Windows Phone

To je tisti del operacijskega sistema, ki je dostopen končnemu uporabniku. V tem delu se nahajajo vse uporabniško dostopne aplikacije - prednameščene in tisti, ki jih je uporabnik namestil sam. Temu delu pripada naša igra Minedefense.

Poleg aplikacij se tukaj nahajajo tudi orodja za upravljanje z operacijskim sistemom. Med ta orodja štejemo upravljalec paketov, ki skrbi za nameščanje in brisanje aplikacij, upravljalec aktivnosti, ki ob zahtevi uporabnika poskrbi, da se zamenja trenutno aktivna aplikacija ter upravljalec virov, ki zapre aplikacijo, če se le-ta začne nepravilno obnašati.

Poglavje 4

Tehnologija XNA

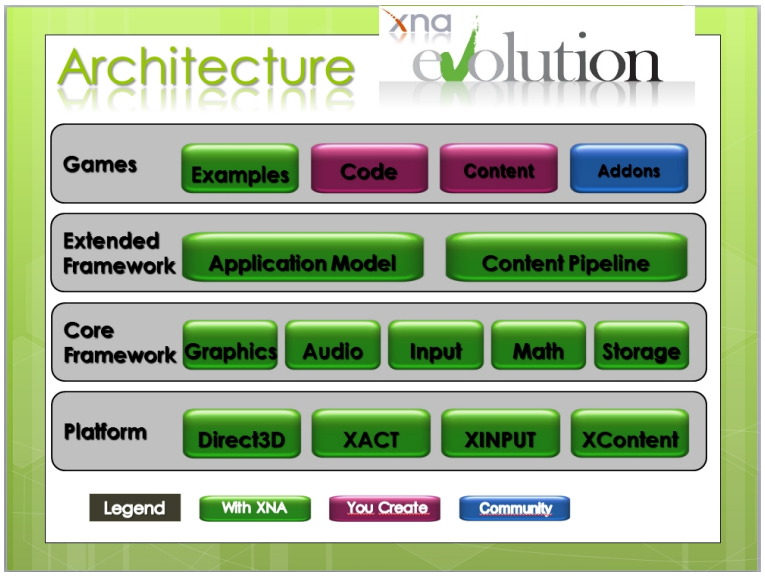
V tem poglavju bomo opisali tehnologijo XNA in knjižnice, ki to tehnologijo implementirajo. Začeli bomo z opisom prvotne knjižnice XNA za Xbox in Windows. Nadaljevali bomo z opisom knjižnice XNI, ki omogoča razvoj iger za sistem iOS. Na koncu bomo opisali še knjižnico MonoGame, ki smo jo uporabili pri razvoju igre Minedefense.

4.1 Opis tehnologije XNA

Kaj sploh je XNA? XNA je programsko ogrodje (knjižnica), ki ga sestavlja set knjižnic .NET. Je poenostavljena različica knjižnice DirectX, namenjena razvoju iger v pogramskem jeziku C# (knjižnica DirectX je namenjena razvoju v C++). Ker v ozadju uporablja tehnologijo .NET in programski jezik C#, so vsi podatki upravljani s strani ogrodja (ang. managed resources), kar peonostavi razvoj iger. Sklop funkcij ogrodja XNA poskrbi, da se razvijalcu ni potrebno ukvarjati z zahtevnimi grafičnimi procesi, kot je to značilno za DirectX.

4.1.1 Plasti ogrodja XNA

Ogrodje XNA sestavljajo 4 plasti [21] (slika 4.1):



Slika 4.1: 4-plastna arhitektura XNA [21].

Platforma

To je najnižja plast ogrodja. Sestavljajo jo osnovne knjižnice, med katere spadajo Direct3D, XACT, XInput in XContent.

Osnovno ogrodje

V tem delu se nahajajo funkcionalnosti za grafiko, zvok, branje vhodnih podatkov, dostop do pomnilnika ter matematične funkcije.

Razširjeno ogrodje

To plast razdelimo na 2 dela: aplikacijski model in cevovod vsebine. Cevovod vsebine pretvori grafično in zvočno vsebino, ter pisave, v format, ki ga razume ogrodje XNA.

Igra

Zadnjo, najvišjo plast, sestavljajo končni izdelki, ki uporabljajo ogrodje XNA. Temu delu pripada naša igra Minedefense.

4.1.2 Zanka igre pri uporabi XNA

Časovno gledano imajo igre, ki bazirajo na tehnologiji XNA (vidno na sliki 4.3), zgradbo, ki je značilna za vse video igre [22] (vidno na sliki 4.2):

- inicializacija komponent,
- nalaganje vsebine v pomnilnik,
- ponavljanje glavne zanke,
- odstranjevanje vsebine iz pomnilnika.

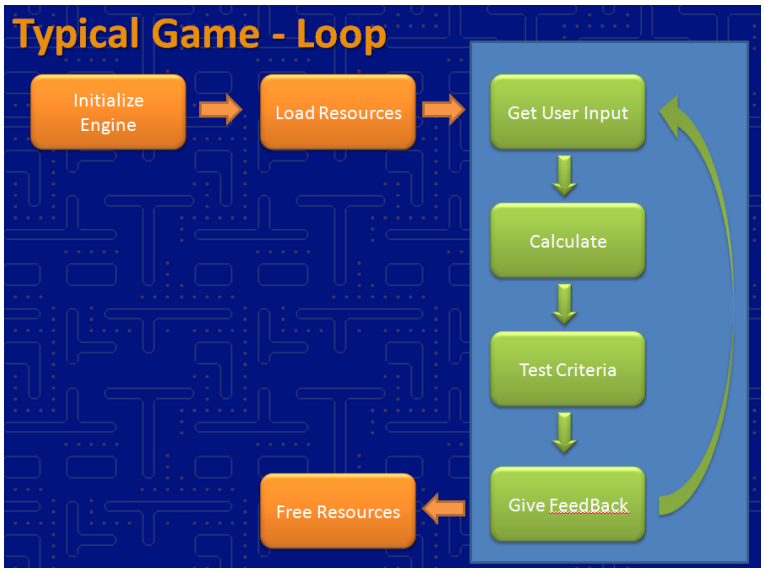
Tudi glavna zanka XNA igre je podobna kot pri video igrah, ki bazirajo na tehnologijah DirectX ali OpenGL. Glavno zanko običajne video igre razdelimo na 4 dele:

- branje vhodnih podatkov,
- računanje novih pozicij in statusa igralcev,
- pregled trkov in statusa igralcev,
- izris preračunanih podatkov na izhodno napravo.

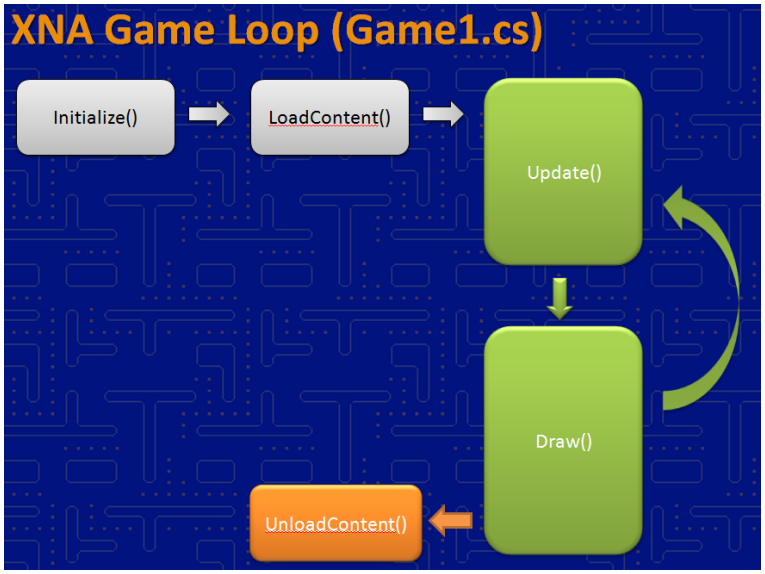
Pri uporabi knjižnice XNA se branje podatkov, računanje pozicij in pregled trkov izvajajo znotraj metode Update. Izris podatkov se izvaja v metodi Draw.

4.2 Knjižnica XNA

Knjižnico XNA [23] so razvili in leta 2004 prvič izdali pri podjetju Microsoft. Glavni namen knjižnice je poenostavitev razvoja preprostih iger, s čimer je knjižnica veliko pripomogla k uspehu samostojnih razvijalcev iger (ang. indie game developer). Igre, razvite z uporabo knjižnice XNA, so na voljo za osebne računalnike z operacijskim sistemom Windows, mobilne naprave



Slika 4.2: Glavna zanka običajne igre [22].



Slika 4.3: Glavna zanka XNA igre [22].

z operacijskim sistemom Windows Phone in igralno konzolo Xbox 360. Samostojni razvijalci video iger lahko igre, razvite s knjižnico XNA, prodajajo preko spletnega mesta Xbox Live Indie Games. Za razvoj iger s knjižnico XNA se uporabljata programski jezik C# in razvojno orodje Visual Studio.

Tehnično knjižnica XNA bazira na ogrodju .NET in za komunikacijo s strojno opremo uporablja knjižnico DirectX. Tako kot knjižnica DirectX, tudi XNA omogoča izrisovanje 2D in 3D grafike.

Različice knjižnice XNA:

- **XNA Game Studio Express** je bila prva različica knjižnice XNA, izdana 11. septembra 2006. Namenjena je bila študentom in razvijalce, ki jim je razvoj iger hobi.
- **XNA Game Studio 2.0**, izdana 13. decembra 2007, je dodala podporo za Xbox Live.
- **XNA Game Studio 3.0**, izdana 30. oktobra 2008, je dodala podporo za LINQ (ang. Language Integrated Query).
- **XNA Game Studio 3.1**, izdana 11. junija 2009, je dodala podporo predvajanju video vsebin v igrah in podporo uporabi Xbox Live avatarjev.
- **XNA Game Studio 4.0**, izdana 16. septembra 2010, je dodala podporo operacijskemu sistemu Windows Phone in s tem tudi podporo zaslonu na dotik.

4.3 Knjižnica XNI

XNI [24] je implementacija tehnologije XNA za operacijski sistem iOS. Za razvoj iger, ki uporabljajo knjižnico XNI, se uporabljata programski jezik Objective-C in razvojno orodje Xcode. Knjižnico razvija Matej Jan, poznan tudi pod psevdonimom Retronator. Trenutno XNI še ne implementira vseh

funkcionalnosti knjižnice XNA, saj je še v fazi razvoja. Kljub temu je knjižnica že zrela za uporabo, v kar prepričuje tudi uspešna izdaja komercialne igre Monkey Labour, ki bazira na knjižnici XNI. XNI smo spoznali tudi na Fakulteti za računalništvo in informatiko. Pri predmetu Tehnologija iger in navidezna resničnost je bilo izdelanih vrsto iger za operacijski sistem iOS.

4.4 Knjižnica MonoGame

MonoGame [25] je odprtokodna implementacija tehnologije XNA. Prva različica je bila izdana leta 2009. Trenutna različica je 3.0.1 in implementira XNA 4.0. Glavni namen knjižnice je uporaba iger, razvitih v XNA za operacijski sistem Windows in igralno konzolo Xbox 360, tudi na ostalih platformah. Trenutno podprte platforme so Windows, Windows Phone, MacOS, iOS, PlayStation Mobile in Linux. Za razvoj iger z uporabo knjižnice MonoGame se uporablja programski jezik C#. Razvijalec lahko uporabi razvojno orodje Visual Studio ali razvojno orodje MonoDevelop.

Podobno kot knjižnica XNA bazira na ogrodju .NET, bazira knjižnica MonoGame na ogrodju Mono. Ogrodje Mono je odprtokodna implementacija ogrodja .NET. Za komunikacijo s strojno opremo MonoGame uporablja grafično knjižnico OpenGL. Ker gre za dejansko implementacijo knjižnice XNA, je tudi zgradba igre, razvite z uporabo knjižnice MonoGame enaka, kot zgradba igre, razvite z uporabo knjižnice XNA.

Zgodovina knjižnice MonoGame:

- **XnaTouch 0.7** je prva različica knjižnice XnaTouch (danes MonoGame), izdana 2. decembra 2009. Implementirala je 2D grafiko, zvok in mrežno podporo. Zaslon na dotik in merilnik pospeška še nista bila povsem podprta. Podpira operacijski sistem iOS.

- **MonoGame 2.0**, izdana 27. oktorba 2011, je bila prva različica z imenom MonoGame in izvira iz XnaTouch različice 0.7.

- **MonoGame 2.1**, izdana 8. decembra 2011, je dodala podporo siste-

mom MacOS, Windows in Linux. Dodana je bila tudi podpora igralnemu ploščku

- **MonoGame 2.5**, izdana 27. marca 2012.
- **MonoGame 2.5.1**, izdana 19. oktobra 2012.
- **MonoGame 3.0**, izdana 21. januarja 2013, je pomemben skok v knjižnici. S to različico je knjižnica pridobila podporo 3D grafike. Prvič je knjižnica na voljo tudi za igralne konzole - PlayStation Mobile.
- **MonoGame 3.0.1**, izdana 3. marca 2013.

Knjižnico MonoGame smo uporabili za razvoj igre Minedefense. Z izdajo operacijskega sistema Windows Phone 8 je namreč podjetje Microsoft ukinilo nadaljnji razvoj knjižnice XNA [26]. Razvoj knjižnice MonoGame pa se nadaljuje in samostojnim razvijalcem še naprej omogoča razvoj preprostih video iger v okolju, ki je na las podobno okolju XNA.

Poglavje 5

Umetna inteligenca iger tower defense

Večino umetne inteligence v igrah tower defense predstavlja iskanje poti sovražnika. Zato bomo v tem poglavju opisali popularne algoritme za preiskovanje prostora v video igrah. Za lažje razumevanje preiskovalnih algoritmov bomo na začetku poglavja opisali tudi načine za predstavitev prostora. Vsebino tega poglavja v grobem povzemamo po knjigi Artificial Intelligence: A Modern Approach (3rd Edition) [27].

5.1 Predstavitev preiskovalnega prostora

5.1.1 Mreže

Dvodimenzionalne mreže so najenostavnejši prikaz sveta. Takšen način prikaza je zelo popularen pri strateških igrah, med katere spada tudi igra Minedefense. Zato smo pri razvoju igre Minedefense za prikaz sveta uporabili dvodimenzionalno mrežo.

Vsako celico mreže označimo kot prehodno (v našem primeru je to pot, ki jo mora sovražnik prehoditi) ali neprehodno (v našem primeru je to območje za gradnjo stolpov). Vsak objekt v igri lahko zavzame eno ali več celic. Kljub temu, da današnje strateške igre uporabljajo tudi tretjo dimenzijo, je

prikaz sveta z dvodimenzionalno mrežo še vedno uporaben. Lahko namreč računamo s tem, da ima ozemlje igre le eno višino, po kateri se lahko agenti sprehajajo. V primeru mostu čez reko bi to pomenilo, da se upošteva samo ena pot: po mostu, ali pod njim.

Mreže so enostavne za implementirati in tudi enostavne za uporabo. Brez dodatnega računanja lahko enostavno ugotovimo tip celice, na kateri se nek objekt nahaja (v naši igri je to lahko pot, ali območje za gradnjo stolpov). Enostavno lahko dostopamo iz poljubne celice do vseh njenih sosed. Mreže delujejo dobro takrat, ko so objekti lahko poravnani s celicami, kar je tipično za strateške igre.

5.1.2 Grafi smernih točk

Graf smernih točk nam poda povezave v svetu, ki so varne za prehod. Smerne točke predstavljajo vozlišča, povezava med dvema točkama pa predstavlja varno pot. Agentu se tako ni potrebno obremenjati s tem, ali bo morda naletel na oviro.

Graf smernih točk je pomnilniško bolj potraten kot dvodimenzionalna mreža. Hraniti moramo namreč dodatne informacije za evidenco o sosednjih celicah, ki jih pri mreži ne potrebujemo. Za naravnejši prikaz gibanja lahko vozlišča hranijo tudi informacije o širini prehoda. Tako agent ni omejen na strogo linijo, saj se lahko naključno giblje znotraj določene širine. Drugi način je pristop ohlapnega sledenja. Pri ohlapnem sledenju moramo upoštevati tudi detekcijo trkov, ki agenta obvarujejo pred trki z ovirami. Takšen način je popularen v prvoosebnih igrah.

5.1.3 Navigacijska omrežja

Navigacijsko omrežje je sestavljeno iz konveksnih poligonov. 2 točki znotraj istega poligona lahko povežemo brez prehoda skozi rob poligona. Vse poti znotraj poligona predstavljajo varne poti. Rob poligona je lahko deljen s sosednjim poligonom, in predstavlja varno pot med poligoni, ali pa z nobenim,

in predstavlja neprehodnost.

Problem grafov smernih točk s pristopom ohlapnega sledenja je detekcija trkov. S številom agentov cena detekcije narašča. Pri navigacijskih omrežjih pa lahko testiramo samo prehod skozi dvodimenzionalni rob poligona in s tem poenostavimo detekcijo trkov.

5.2 Algoritmi za preiskovanje sveta

5.2.1 Iskanje v globino

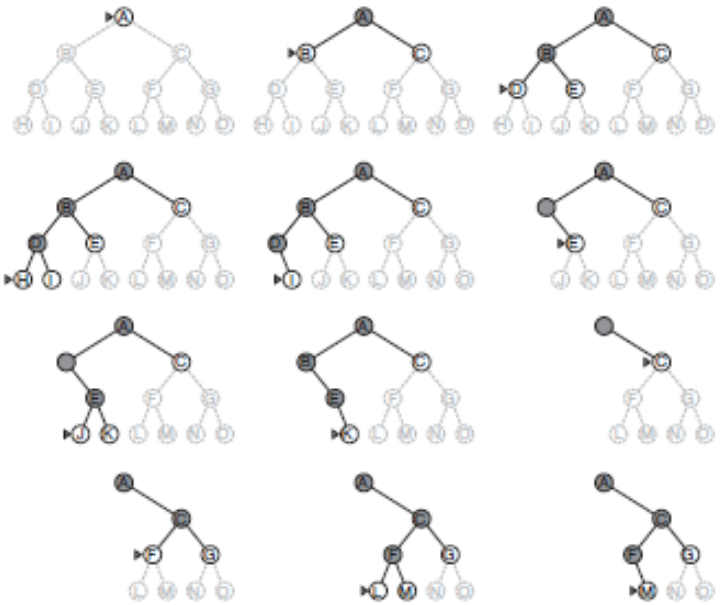
Iskanje se takoj poda v najglobji del sveta, dokler ne pride v vozlišče, ki nima naslednika. Iskanje se izvaja po principu LIFO (ang. last in first out), saj se nazadnje obiskano vozlišče uporabi kot izhodišče za iskanje novega vozlišča. Naslednje vozlišče je zagotovo najglobje, saj je za en nivo nižji od starša, ki je bil pred tem znan kot najglobje vozlišče. Vsa preiskana vozlišča, ki nimajo nepreiskanih potomcev izbrišemo iz spomina. Delovanje algoritma si lahko ogledamo na sliki 5.1.

Če se nahajamo v končnem svetu, potem je takšen algoritem popoln, saj bo slej ali prej preiskal vsa vozlišča. V primeru neskončnega sveta pa lahko naletimo na naskončno pot, ki je takšen algoritem ne zna preiskati.

5.2.2 Iskanje v širino

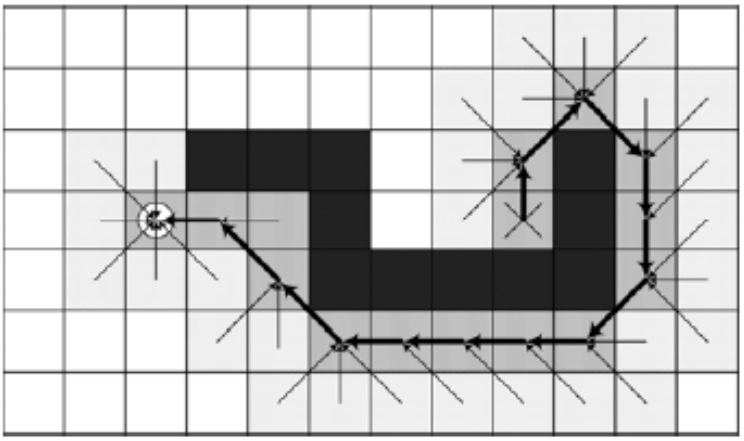
Pri iskanju v širino najprej preiščemo korensko vozlišče. Nato preiščemo vse potomce korenskega vozlišča, nato potomce potomcev in tako naprej. Za množico vozlišč se uporablja FIFO (ang. first in first out). Nova vozlišča gredo vedno na konec vrste, stara vozlišča, ki imajo nižjo globino, pa raziščemo naprej. Takšen princip je razviden iz slike 5.2

Algoritem je popoln. Če se iskano vozlišče nahaja na končni globini, ga bo algoritem našel. Pri tem predpostavljamo, da je vejitveni faktor končen. Algoritem je optimalen, če je cena vseh poti enaka.



Slika 5.1: Iskanje v globino [27].

Slika 5.2: Iskanje v širino [27].



Slika 5.3: Neoptimalna pot algoritma najprej najboljši [27].

5.2.3 Najprej najboljši

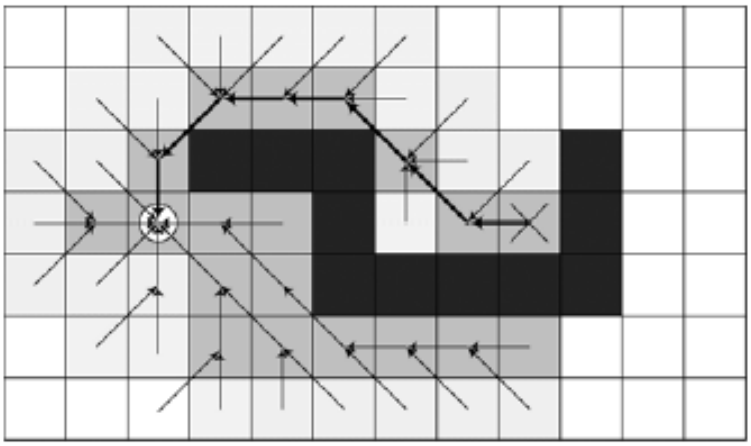
Algoritem najprej najboljši spada med informirane metode in za preiskovanje uporablja algoritem najprej v širino. Pri tem izračuna razdaljo vsakega vozlišča do cilja in naredi prioritetno vrsto glede na izračunane razdalje.

Algoritem sicer vedno najde rešitev, ni pa nujno, da je ta rešitev tudi optimalna. Lahko se namreč zgodi, da se že na začetku preiskovanja usmeri v napačno smer (slika 5.3). Če po tej smeri vseeno najde pot, se algoritem konča, dobljena pot pa ni najboljša.

5.2.4 Dijkstrin algoritem

Tudi Dijkstrin algoritem v osnovi uporablja preiskovanje v širino. Pri tem vodi evidenco o ceni od začetnega do danega vozlišča. Tako lahko za nadaljevanje iskanja vedno izbere tisto vozlišče, ki doda najmanjšo ceno.

Bistveno prednost algoritma predstavljajo različno utežene regije sveta. Te uteži lahko pomagajo agentu, da se izogne delom sveta, kamor ne sme priti. Z utežmi lahko naprimer določamo tveganje za prehod skozi določen del sveta. Dijkstrin algoritem mora zato vsakič znova preveriti, ali je nova pot do danega vozlišča cenejša, kot prejšnja.



Slika 5.4: Optimalna pot algoritma A* [27].

Algoritem je popoln v primeru, ko je cena vsakega vozlišča večja od majhne konstante. Če imajo vozlišča ceno 0, se namreč lahko zgodi, da se algoritem ujame v neskončno zanko.

5.2.5 A*

Algoritem A* reši večino problemov prej opisanih algoritmov, pri tem pa porabi manj spomina in potrebuje veliko manj časa.

Ob predpostavki, da uporabljamo sprejemljivo hevristično funkcijo, algoritem vedno najde optimalno rešitev (slika 5.4). Sprejemljiva hevristična funkcija nikoli ne preceni dejanske cene poti do cilja (za vsako vozlišče n in vsakega njegovega naslednika n' velja, da ocena cene iz vozlišča n do cilja ni nič večja kot cena koraka iz n v n' plus ocenjena cena iz n' do cilja).

A* torej združuje algoritem najprej najboljši in Dijkstrin algoritem, saj pri izračunu cene upošteva tako dano ceno kot hevristično ceno. Dana cena je dejanska cena od začetnega do trenutnega vozlišča (Dijkstrin algoritem). Hevristična cena pa je ocenjena cena od trenutnega vozlišča do cilja (algoritem najprej najboljši). A* ima torej množico sortirano po končni ceni, ki jo izračunamo s pomočjo enačbe *končna cena = dana cena + (hevristična cena*

** hevristična utež).*

Obstajajo tudi algoritmi, ki dopolnjujejo algoritem A*:

- A* z iterativnim poglabljanjem (ang. iterative deepening A* - IDA*).
- Rekurzivni najprej najboljši (ang. recursive best first search - RBFS).
- Prostorsko omejen A* (ang. memory bounded A* - MA*).

Poglavje 6

Uporaba XNA v razvoju igre Minedefense

Na začetku poglavja so opisani osnovni razredi in metode, ki sestavljajo knjižnico XNA. Sledi opis glavne zanke igre Minedefense in nato razlaga posameznih delov igre.

6.1 Osnovni razredi in metode knjižnice XNA

Initialize

Metoda *Initialize* se uporablja za inicializacijo začetnih vrednosti objekta. Je prva metoda, ki se kliče po konstrukciji objekta.

LoadContent

LoadContent je metoda, ki se uporablja za nalaganje teksture, pisav in zvočnih efektov. Klic metode *LoadContent* se praviloma izvrši po klicu metode *Initialize*. Nalaganje vsebine nam omogoča razred *ContentManager*.

Update

Metoda *Update* se kliče v vsakem ciklu glavne zanke igre. Praviloma se znotraj te metode procesira vhodne podatke (preko razreda *TouchLo-*

cation), umetno inteligenco in pravila igre.

Draw

Podatke, ki smo jih izračunali v metodi *Update*, je potrebno prikazati na zaslonu. To nam omogoča metoda *Draw*. V tej metodi z uporabo razreda *SpriteBatch* izrisujemo teksture.

UnloadContent

Čeprav zna smetar iz pomnilnika odstraniti nerabljene objekte, je v nekaterih primerih to potrebno opraviti ročno. To storimo v metodi *UnloadContent*, ki se kliče ob destrukciji objekta.

SpriteBatch

Razred *SpriteBatch* se uporablja za izris tekstur na zaslon. Objekt tipa *SpriteBatch* praviloma inicializiramo ob inicializaciji igre. Konstruktorju moramo podati objekt *GraphicsDevice*, preko katerega nastavimo parametre zaslona.

Izris teksture izvršimo s klicem metode *Draw*. Obstaja več implementacij te metode, v naši igri je bila največkrat uporabljena različica *Draw(texture, position, sourceRectange, color)*. Parametri pomenijo sledeče: *texture* je tekstura za izris, *position* je pozicija, kamor se bo tekstura izrisala, *sourceRectangle* je del teksture, ki bo izrisan na zaslon, *color* je barva odtenka. Paziti moramo, da metodo *Draw* vedno kličemo znotraj metode *Draw* naše igre. Pred klicem metode *Draw* je potrebno na objektu poklicati metodo *Begin*. Risanje zaključimo z metodo *End*, ki v predpomnilnik naložene slike dejansko izriše na zaslon.

ContentManager

ContentManager je razred, ki nam omogoča nalaganje vsebine. To se praviloma izvaja v metodi *LoadContent*. Naložimo lahko teksture, zvočne datoteke in pisave. Objekta tipa *ContentManager* nam ni potrebno ustvariti, saj za to poskrbi sama igra. Temu objektu je ime *Content*.

Za nalaganje vsebine uporabimo metodo *Load<T>(assetName)*. *T* predstavlja vrsto vsebine, ki jo bomo naložili. Atribut *assetName* je ime datoteke, iz katere se bo vsebina prebrala.

TouchPanel

TouchPanel je razred, preko katerega dostopamo do vhodnih podatkov zaslona na dotik. Objekt tega razreda se ob inicializaciji igre inicializira samodejno.

Sam objekt *TouchPanel* še ne nosi nikakršnih vhodnih podatkov o dotikih. Za pridobitev podatkov je potrebno poklicati metodo *GetState()*, ki vrne objekt tipa *TouchCollection*. Ta vsebuje vse dotike, ki se trenutno prisotni na zaslonu. Ker je igra Minedefense zasnovana kot igra, ki se jo upravlja z enim samim dotikom, vedno vzamemo samo prvi objekt iz seznama *TouchCollection*. Pridobljen objekt je tipa *TouchLocation* in hrani lokacijo in stanje dotika. Vsi našteti klici metod se praviloma vršijo v metodi *Update*.

SoundEffect

Razred *SoundEffect* se uporablja za hranjenje in predvajanje zvoka. Zvok naložimo z objektom *ContentManager*. Za predvajanje naloženega zvoka pokličemo metodo *Play()*.

Texture2D

Razred *Texture2D* uporabljamo za hrambo tekstur. V kodi igre Minedefense ne koristimo nobene metode razreda. Teksture naložimo preko objekta *ContentManager*.

SpriteFont

Podobno kot velja za razred *Texture2D*, velja tudi za razred *SpriteFont*. Objekte tega tipa uporabljamo za hranjenje pisave.

6.2 Glavna zanka igre

Glavna zanka igre se izvaja v razredu Game1, ki je izpeljava razreda Game. Ko se ob zagonu igre ustvari objekt tipa Game1, se pokličeta metodi Initialize in LoadContent. Nato sledi izvajanje neskončne zanke, znotraj katere se ponavljajo klici metod Update in Draw. Ko uporabnik izbere izhod iz igre, se neskončna zanka prekine, pokliče se metoda UnloadContent in igra se zapre.

Initialize

V metodi *Initialize* se inicializirajo vse scene menija. Te scene so *_mainMenu* (objekt tipa *MainMenu*), *_helpMenu* (objekt tipa *HelpMenu*) in *_aboutMenu* (objekt tipa *AboutMenu*). Skozi potek dogodkov se scene menija ne spreminjajo, zato jih lahko inicializiramo ob inicializaciji igre. Poleg inicializacije scen se nastavijo tudi vrednosti objekta *GraphicsDevice*.

LoadContent

Podobno kot deluje metoda *Initialize* deluje tudi metoda *LoadContent*. V metodi *LoadContent* se inicializira objekt *_spriteBatch*. Nato se kličejo metode *LoadContent* vseh scen, ki so se inicializirale v metodi *Initialize*. Tak pristop nam omogoča, da vsaka scena naloži vsebino, ki jo sama potrebuje.

Update

Metoda *Update* se kliče v vsakem ciklu glavne zanke igre. Za to poskrbi arhitektura XNA. Najprej se iz objekta *TouchPanel* prebere trenutna vrednost dotika, ki se shrani v objekt *_touchLocation*. Branje dotika je potrebno implementirati tako, da se ob vsakem obhodu zanke le-ta prebere samo enkrat. V nasprotnem primeru bi lahko prišlo do različnih vrednosti po vsakem branju. Nato se v odvisnosti od stanja igre, ki je shranjeno v objektu *_gameState*, posodobi ustrezna scena.

Draw

Draw deluje podobno kot metoda *Update*. Glede na stanje, zapisano v objektu *_gameState*, se na ustrezni sceni pokliče metoda *Draw*. Pred tem je potrebno počistiti vsebino zaslona in objektu *_spriteBatch* povedati, da bomo izrisovali vsebino na zaslon. Zaslon počistimo s klicom metode *Clear* objekta *GraphicsDevice*. Izrisovanje pričnemo s klicem metode *Begin* objekta *_spriteBatch*. Ko smo poklicali vse potrebne metode *Draw*, izrisovanje zaključimo s klicem metode *End* objekta *_spriteBatch*.

UnloadContent

Metoda *UnloadContent* je v naši igri prazna, saj ne uporabljamo vsebine, ki je smetar ne bi znal odstraniti.

Preklapljanje scen

V prid hitrosti izvajanja se v odvisnosti od stanja igre vedno posodablja in izrisuje samo aktivna scena. Za potrebe spreminjanja stanja smo implementirali 3 javne metode, ki so dostopne vsem scenam.

Najpreprostejša je metoda *SetGameState(gameState)*. Metoda prejme 1 parameter, ki je vrednost novega stanja. Ta vrednost se shrani v objekt *_gameState*.

StartNewGame() je metoda, ki se uporablja za inicializacijo scene *_gameLogic* (objekt tipa *GameLogic*). Za preklop med različnimi stanji menija je zgoraj opisana metoda *SetGameState* zadostovala. Pri vstopu v novo igro pa je potrebno najprej ročno odstraniti prejšnji objekt in nato inicializirati novega. Na koncu je potrebno nastaviti še ustrezno stanje igre.

Metoda *ResumeGame* nastavi sceno *_gameLogic* na aktivno in ob tem obdrži njene stare vrednosti. Ta metoda se uporablja v primeru, ko igralec iz igre vstopi v glavni meni, nato pa se želi vrniti v samo igro.

6.3 Scene in stanja igre

Kot smo opisali že v poglavju 6.2 se igra lahko nahaja v različnih stanjih. Za različna stanja se uporabljajo različne scene. Razredi, ki predstavljajo posamezne scene so MainMenu, AboutMenu, HelpMenu in GameLogic. Vsi našteti razredi dedujejo iz razreda BaseComponent, ki vsebuje osnovne funkcionalnosti za delo s posamezno sceno.

6.3.1 Scene menija

Meni se lahko nahaja v 3 različnih stanjih, vsako stanje je predstavljeno s posamezno sceno.

MainMenu

Razred *MainMenu* predstavlja sceno glavnega menija. V tem delu se nahajajo gumbi *New game* (zažene novo igro), *Resume game* (nadaljuje prejšnjo igro), *About* (prikaže podatke o igri) in *Help* (prikaže navodila za igranje). Na vsak gumb je vezan poslušalec, ki se proži ob kliku. Poslušalec pokliče ustrezno metodo razreda *Game1* za preklop stanja igre.

AboutMenu

Razred *AboutMenu* predstavlja sceno menija, ki prikazuje podatke o igri. V sceni *AboutMenu* se nahaja gumb *Menu*, ki igralca preusmeri na glavni meni.

HelpMenu

Razred *HelpMenu* predstavlja sceno menija, ki prikazuje navodila za igranje. Je skoraj identičen razredu *AboutMenu*. Razreda se razlikujeta samo po besedilu, ki se prikaže na sceni.

Delovanje scen menija:

LoadContent

V metodi *LoadContent* se naloži tekstura ozadja in inicializirajo se vsi gumbi. Pri tem se gumbom pripne tudi ustrezne poslušalce.

Update in Draw

V metodah Update in *Draw* se kličejo istoimenske metode vseh gumbov.

Poslušalci

Za vsak gumb na sceni obstaja en poslušalec, ki se proži ob kliku na gumb. Ob proženju kliče ustrezno metodo za preklop stanja, ki se nahaja v razredu *Game1*.

6.3.2 Scena igre

Scena igre je predstavljena z razredom *GameLogic*. Inicializira se ob kliku na gumb *New game*.

Initialize in LoadContent

Ob inicializaciji scene se inicializirajo vsi upravljalci igre (ang. controller). Pri tem se nastavijo tudi medsebojne reference upravljalcev, da lahko komunicirajo drug z drugim. V metodi *LoadContent* se pokličeje istoimenske metode upravljalcev. Ti morajo sami poskrbeti za nalaganje potrebne vsebine.

Update

V metodi Update se preverijo pravila igre, kar lahko razberemo iz kode 6.1. Če je igralec izgubil vsa življenja se prikaže okno z napisom *Game over* in igra se zaključi. Igra se zključi tudi v primeru, ko sovražnik nima več enot za napad. V tem primeru se prikaže obvestilo *Victory*. Če nobeden od pogojev za konec igre ni izpolnjen, se igra lahko nadaljuje. Nadaljevanje proži klic metod *Update* posameznih upravljalcev

Draw

Tukaj se kličejo metode *Draw* posameznih upravljalcev.

Koda 6.1: Preverjanje pravil igre.

```
1 // If player has lost all of his lives, the game is over.
```

```
2  if (_player.PlayerLives <= 0)
3  {
4      _popupHud.ShowPopup("Game over", new Vector2(120, 45));
5      return;
6  }
7
8  // If there are no more creep waves, the player has won.
9  if (_waveManager.EndOfWaves)
10 {
11     _popupHud.ShowPopup("Victory", new Vector2(135, 45));
12     return;
13 }
```

6.4 Predstavitev entitet

6.4.1 Sprite

Vse entitete na sceni so predstavljene z razredom *Sprite*. Objekt tipa *Sprite* hrani teksturo (*_texture*), velikost (*_size*), pozicijo (*_position*), rotacijo (*_rotation*), hitrost gibanja (*_velocity*) in središče entitete (*_center*).

Sprite

Razred *Sprite* zaradi narave inicializacije ne vsebuje metod *Initialize* in *LoadContent*. Entitete se namreč ne naložijo ob inicializaciji igre, ampak se dinamično dodajajo skozi igro. Zato je bolj primerna inicializacija s kontruktorjem. V razredu obstajata 2 implementaciji konstruktorja. Parametri konstruktorja *Sprite(texture, position, size)* pomenijo naslednje: *texture* je tekstura entitete, *position* je začetna pozicija entitete, *size* je velikost entitete. Konstruktor *Sprite(texture, position)* predpostavlja, da je velikost entitete enaka velikosti teksture.

Draw

Metoda *Draw* izriše teksturo entitete na njeno pozicijo. Osnovna metoda je *Draw(spriteBatch)*. Ta predpostavlja, da se sličica za izris prične na točki 0 teksture. V večini primerov ta predpostavka drži. Nekatere entite v naši igri pa so animirane in potrebujejo za vsak okvir

animacije drugačno sličico. V tem primeru moramo uporabiti metodo *Draw(spriteBatch, textureOffset)*, kjer parameter *textureOffset* pomeni številko sličice v teksturi.

6.4.2 Button

Razred *Button* se uporablja za predstavitev gumbov v igri. Vsak gumb lahko obravnavamo kot entito, s katero lahko uporabnik interaktira. Zato razred *Button* razširja razred *Sprite* in dodaja funkcionalnosti za interakcijo. Nova objekta sta *_bounds* in *_state*. Prvi predstavlja območje, kjer bo igra zaznala dotik, drugi pa trenutno stanje gumba.

Update

V metodi *Update* se preveri, ali se lokacija dotika nahaja v območju, ki ga pokriva gumb (*_bounds*). Če se dotik res nahaja v območju gumba, se preveri še vrsta dotika. Razred *Button* razlikuje med dvema vrstama dotika: pritisnjeno (ang. pressed) in spuščeno (ang. released). Ko se določi vrsta dotika, lahko metoda proži ustreznega poslušalca.

Draw

Metoda *Draw* izriše teksturo gumba na njegovo pozicijo. Usrezna slika teksture se določi na podlagi stanja gumba.

Poslušalci

V opisu metode *Update* smo omenili proženje poslušalcev. V razredu *Button* uporabljamo 2 poslušalca. Poslušalec *Pressed* se proži ob pritisku na gumb, poslušalec *Released* pa takrat, ko igralec spusti gumb.

6.4.3 Creep

Sovražnikove enote so predstavljene z razredom *Creep*. Razred *Creep* deduje iz razreda *Sprite* in funkcionalnosti za delo s sovražnikovimi enotami. Pomembni objekti so *_damageType*, *_currentHealth* in *_speed*. Prvi hrani tip

stolpa, ki lahko napada to enoto, drugi hrani trenutno zdravje enote, tretji pa predstavlja hitrost gibanja.

Update

V metodi *Update* se najprej preveri, če je enota že dosegla konec poti (seznam _*waypoints* je prazen). Če je cilj dosežen, se enoto uniči, izvajanje metode pa se zaključi. Če cilj ni dosežen, je potrebno izračunati novo pozicijo enote. Ko je enota dovolj blizu naslednjega polja v seznamu _*waypoints*, se to polje uporabi za novo pozicijo enote. V tem primeru je potrebno ponovno izračunati smer enote. Za izračun smeri se uporabi XNA funkcija *Atan2(y, x)*. Če pa je enota od naslednjega polja preveč oddaljena, in ga v tem ciklu zanke še ne doseže, je potrebno novo pozicijo izračunati na podlagi stare pozicije, smeri gibanja in hitrosti. Delovanje metode lahko razberemo iz kode 6.2.

Draw

Gibanje sovražnikove enote je animirano. Hitrost animacije je odvisna od hitrosti gibanja. V metodi *Draw* moramo zato izračunati odmik trenutne sličice animacije glede na čas, ki je potekel od zadnje spremembe animacije in hitrosti gibanja. Ko izračunamo odmik, na zaslon izrišemo ustrezno sličico.

SetWayPoints

Metoda *SetWaypoints(waypoints)* nastavi podani seznam točk kot izračunano pot, ki jo mora enota prehoditi. Pri tem je potrebno paziti, da ne shranimo samo reference, ampak je potrebno narediti globoko kopijo (ang. deep copy) seznama. Kasneje bomo namreč zapise seznama postopoma odstranjevali in brez globoke kopije bi s tem praznili originalni seznam.

Koda 6.2: Simulacija gibanja sovražnikove enote.

```
1 // If there are no more waypoints, kill the creep, because it has reached the end of the
      map.
2 if (_waypoints.Count == 0)
```

```
3  {
4      _alive = false;
5      return;
6  }
7
8  // If the creep is close enough to the next waypoint, simply set its position to next
       waypoint.
9  if (DistanceToDestination < _speed)
10 {
11     _position = _waypoints.Dequeue();
12
13     if (_waypoints.Count > 0)
14     {
15         Vector2 direction = _waypoints.Peek() - _position;
16         direction.Normalize();
17         _rotation = (float)Math.Atan2(direction.X, -direction.Y);
18     }
19 }
20
21 // But if creep is too far away from next waypoint, calculate its new position using its
       direction and speed.
22 else
23 {
24     Vector2 direction = _waypoints.Peek() - _position;
25     direction.Normalize();
26
27     _velocity = Vector2.Multiply(direction, _speed);
28
29     _position += _velocity;
30 }
```

6.4.4 Tower in TowerTypes

Razred *Tower* predstavlja stolp, ki ga igralec zgradi za obrambo pred sovražnikovimi enotami. Deduje iz razreda *Sprite*. Dodane lastnosti hranijo ceno stolpa (_ *costAmount* in _ *materialType*), vzdržljivost (_ *durability*), tip enot, ki jih lahko stolp napada (_ *damageType*), količino škode, ki jo povzroči en napad (_ *damage*), hitrost napada (_ *cooldown*) in domet stolpa (_ *radius*).

Tower je zgolj osnovni razred. Različni tipi stolpov so predstavljeni z različnimi razredi, ki vsi razširjajo razred *Tower* - so otroci razreda *Tower*. Ti razredi se nahajajo v razredu *TowerTypes*.

Update

V metodi *Update* se najprej preveri, če stolp že ima določeno tarčo, oziroma če je tarča znotraj dometa. Če vsaj eden od pogojev ni izpolnjen, se pokliče metoda *FindClosestEnemy*, ki poišče najbližjo tarčo. Nato je stolp potrebno usmeriti proti nastavljeni tarči. To se zgodi ob klicu metode *FaceTarget*. Ko je tarča določena in stolp usmerjen proti njej, se pokliče metoda *ShootTarget*, ki proti tarči izstreli izstrelek.

Draw

Metoda *Draw* najprej preveri, če je stolp v stanju napada (ima določeno tarčo). Ko stolp miruje, ni animacije, zato se izriše sličica stolpa v mirovanju. Ko stolp napada, pa je potrebno prikazati animacijo. Trenutna sličica animacije se izračuna na podlagi časa, ki je potekel od prejšnje sličice animacije in hitrosti napada.

FindClosestEnemy

Metoda *FindClosestEnemy* se sprehodi skozi vse sovražnikove enote, ki so trenutno na sceni, in izbere tisto, ki je stolpu najbližja. Če so vse enote oddaljene več, kot je domet stolpa, se stolpu ne določi tarče.

FaceTarget

Metodo *FaceTarget* (koda 6.3) smo omenili že v opisu metode *Update*. Metoda usmeri stolp proti njegovi tarči. Za izračun smeri se uporabi XNA funkcija *Atan2(y, x)*. Če stolp nima tarče, se usmeri proti vrhu.

ShootTarget

V metodi *ShootTarget* se vrši napad stolpa. Napad se lahko prične šele takrat, ko je od prejšnjega napada preteklo dovolj časa (potreben čas je shranjen v objektu *_cooldown*). Predno stolp dejansko izstreli izstrelek proti tarči, je potrebno napad animirati. Ko se animacija napada odvije do konca, lahko dodamo nov objekt tipa *Bullet* in ga usmerimo proti tarči. Pri tem se stolpu zmanjša vzdržjivost (*_durability*) in ponastavi čas napada.

Koda 6.3: Usmerjanje stolpa proti tarči.

```
1 Vector2 direction = _center - _target.Center;
2 direction.Normalize();
3
4 _rotation = (float)Math.Atan2(-direction.X, direction.Y);
```

6.4.5 Bullet

Razred *Bullet* se uporablja za prikaz izstrelkov. Deduje iz razreda *Sprite*. Pomembnejši novi objekti so *_damage* (škoda, ki jo izstrelek povzroči), *_target* (tarča izstrelka) in *_speed* (hitrost leta).

Update

V metodi *Update* se najprej določi nova smer izstrelka (s klicem metode *SetDirection*), nato pa se izračuna nova pozicija izstrelka.

SetDirection

V vsakem ciklu zanke igre se sovražnikove enote premaknejo. Ker ne želimo, da izstrelki stolpov grešijo, je potrebno ponovno določiti smer izstrelka. Smer se izračuna v metodi *SetDirection*.

6.5 Ozemlje in iskanje poti

Upravljalec *Map* skrbi za ozemlje igre. Ta upravljalec je predstavljen z razredom *Map*. Vsa logika za računanje poti, ki jo morajo sovražnikove enote prehoditi, se nahaja v razredu *PathFinder*.

6.5.1 Map

Ozemlje igre Minedefense je razdeljeno v matriko 8x8 polj. Programsko je matriko najenostavneje implementirati z dvodimenzionalno tabelo. Takšna tabela je tudi *_terrain*, ki se nahaja v razredu *Map* in predstavlja ozemlje naše igre. Drugi pomemben objekt je seznam polj, ki predstavljajo pot skozi ozemlje. To je objekt *_waypoints*.

Initialize

Tabela *_terrain*, ki predstavlja ozemlje, se zgradi v metodi *Initialize*. Nato se v razredu *PathFinder* izračuna pot skozi ozemlje in shrani v objekt *_waypoints*.

Draw

Metoda *Draw* se sprehodi skozi celotno tabelo *_terrain* in na podlagi vrednosti posameznega polja tabele določi teksturo, ki se bo na lokaciji tega polja izrisala na zaslon. Ko je celotno ozemlje izrisano na zaslon, se pokliče metoda *DrawPortals*.

DrawPortals

V metodi *DrawPortals* se na začetno in končno polje poti dodajo portali. Ti portali so zgolj lepotni dodatek in na samo logiko igre nimajo vpliva.

6.6 Sovražnik

Delovanje posamezne enote sovražnika smo omenili že v poglavju 6.4.3. Več enot skupaj sestavlja eno stopnjo napada (razred *Wave*). Za pravilno razporeditev napadov poskrbi upravljalec *WaveManager*.

6.6.1 Wave in WaveTypes

Razred *Wave* predstavlja eno stopnjo igre. V vsaki stopnji je lahko poljubno število enot, ki jih hranimo v seznamu *_creeps*. Poleg seznama imamo tudi objekte, ki nosijo enake vrednosti kot objekti razreda *Creep* in se uporabljajo zgolj pri inicializaciji novih enot. Pomembnejši objekt je tudi *_spawnCooldown*, ki določa, koliko časa mora preteči med generiranjem (and. spawn) dveh enot.

Razred *Wave* je zgolj osnovni razred. Za posamezne stopnje se uporabljajo različni razredi, ki razširjajo razred *Wave*. Imenujemo jih otroci ra-

zreda *Wave*. Vsak otrok predstavlja natanko eno stopnjo, in vsaka stopnja potrebuje natanko enega otroka. Ti razredi so združeni v razredu *WaveTypes*.

Wave

Razred *Wave* ne uporablja metod *Initialize* in *LoadContent*. Njuno funkcionalnost nadomesti konstruktor. Vsak otrok v konstruktorju nastavi svoje vrednosti.

Update

V metodi *Update* se preveri, če je v tej stopnji potrebno dodati še kakšno enoto. V tem primeru se preveri, če je preteklo dovolj časa od generiranja zadnje enote. Takrat se doda nova enota. Na koncu metode se kličej metoda *UpdateCreeps*.

Draw

Tukaj se skozi zanko izrišejo vse enote.

UpdateCreeps

V metodi *UpdateCreeps* se posodobijo vse enote. Pri tem se za vsako enoto preveri, če je dosegla konec poti oziroma če jo je igralec uspel uničiti - tako delovanje lahko razberemo iz kode 6.4. Če je enota dosegla konce poti, je potrebno igralcu odšteti ustrezno število življenj (metoda *RemoveLives* razreda *Player*). Če je igralec uničil enoto, pa je potrebno igralcu dodati ustrezno število virov (metoda *AddResources* razreda *Player*).

SpawnCreep

Metoda *SpawnCreep* generira novo enoto in jo doda v seznam *_creeps*.

FindAndSetTextures

Metoda *FindAndSetTextures* v seznamu parametra poišče ustrezno teksturo in jo nastavi kot teksturo te stopnje.

Koda 6.4: Preverjanje konca poti sovražnikove enote.

```
1  if (creep.IsDead)
2  {
3      // If the creep has any health, it has reached the end and the player should loose
           one life.
4      if (creep.CurrentHealth > 0)
5      {
6          // If the creep is boss, player should loose all lives and the game is over.
7          if (_isBoss)
8              _player.RemoveLives(_player.PlayerLives);
9          // Else if creep is not boss, player should loose one life.
10         else
11             _player.RemoveLives(1);
12     }
13     // Else if creeps has no life, it was killed by the player and the player should be
           given resources.
14     else
15     {
16         _player.AddResources(_creepBounty, _bountyType);
17     }
18
19     _creeps.Remove(creep);
20     i--;
21 }
```

6.6.2 WaveManager

WaveManager je razred, ki predstavlja upravljalca sovražnikovih napadov. Razred vsebuje seznam vseh stopenj (*_waves*), referenco na trenutno stopnjo (*_currentWave*) in čas, ki mora preteči med koncem ene in začetko nove stopnje (*_timeSinceLastWave*).

Initialize

V metodi *Initiliaze* se kreirajo vse stopnje in se dodajo v seznam *_waves*.

Update

Metoda *Update* (koda 6.5) preveri, če je trenutno kakšna stopnja v teku. Če ni, pokliče metodo *PrepareNextWave*. Če je stopnja sicer pripravljena, ni pa še bila zagnana, se preveri, če je morda preteklo že

dovolj časa, da se lahko prične nova stopnja. V tem primeru se kliče metoda *StartNextWave*. Ko pa je stopnja v teku in je že bila zagnana, se trenutna stopnja posodobi. Po koncu posodobitve se preveri, če se je s to posodobitvijo stopnja končala. Takrat se stopnja odstrani in s tem pridemo v stanje, ko nobena stopnja ni v teku.

PrepareNextWave

PrepareNextWave nastavi naslednjo stopnjo iz seznama *_waves* kot aktivno. Ob tem pokliče tudi metodo *FindAndSetTextures* razreda *Wave*.

StartNextWave

Metoda *StartNextWave* zažene stopnjo, ki smo jo nastavili v metodi *PrepareNextWave*.

Koda 6.5: Posodabljanje trenutne stopnje.

```
1  // If there is no wave, prepare one.
2  if (_currentWave == null)
3      PrepareNextWave();
4
5  // else if wave is not yet finished, update this wave.
6  else if (!_waveFinished)
7  {
8      _currentWave.Update(gameTime);
9      if (_currentWave.RoundOver)
10         _currentWave = null;
11 }
12
13 // Else if wave is finished, check if new one can be started already.
14 else
15 {
16     _timeSinceLastWave -= (float)gameTime.ElapsedGameTime.TotalSeconds;
17     if (_timeSinceLastWave <= 0)
18         StartNextWave();
19 }
```

6.7 Igralec

Razred *Player* predstavlja upravljalca, ki igralcu omogoča interakcijo z igro. Obenem ta upravljalec nadzira tudi delovanje stolpov in hrani podatke o trenutnem stanju igralca. Število življenj se nahaj v objektu *_lives*. Viri, ki jih ima igralec na razpolago, so zapisani v objektu *_resources*. Seznam vseh stolpov in izstrelkov na sceni predstavljata objekta *_towers* in *_bullets*. Tip stolpa, ki ga igralec želi zgraditi, pa je shranjen v objektu *_towerToBuild*.

Initialize

V metodi *Initialize* se nastavi začetno število življenj in virov.

Update

V metodi *Update* se najprej določi celica ozemlja, kjer se nahaja lokacija dotika. Nato se stolp, ki se nahaja v tej celici, nastavi kot trenutno izbran stolp. Če na tej celici ni stolpa, se vrednost izbranega stolpa ne nastavi. Izbiro stolpa omogoča metoda *SelectTowerInCell*. Istočasno se ob spustu dotika v celico (če je ta prazna) doda nov stolp, ki je bil kupljen pred tem. To omogoča metoda *BuildTower*. Na koncu se kličeta metodi *UpdateTowers* in *UpdateBullets*.

Draw

Draw se sprehodi skozi seznam stolpov in skozi seznam izstrelkov ter kliče metodo *Draw* posameznega stolpa oziroma izstrelka.

RemoveLives

To je preprosta javna metoda, ki jo lahko kličejo ostali upravljalci. Igralcu se odstrani v parametru podano število življenj.

AddResources

Tudi *AddResources* je javna metoda. Sprejme 2 parametra: *resources*, ki je število virov in *resourcesType*, ki je tip vira. Igralcu se prišteje določeno število virov.

SetTowerToBuild

SetTowerToBuild je javna metoda. V parametru podamo tip stolpa, ki ga igralec želi zgraditi. Nato se ustvari začasna instanca tega tipa. Če ima igralec dovolj virov za nakup takega stolpa, se tip stolpa shrani v razredno spremenljivko *_towerToBuild*.

DrawTowerPreview

Izvajanje metode *DrawTowerToBuild* se takoj prekine, če igralec nima izbranega stolpa za nakup. V nasprotnem primeru se poišče tekstura stolpa, ki ga igralec želi zgraditi. Ta tekstura se nato izriše na trenutno lokacijo dotika. Ob tem se poveča tudi prosojnost teksture, saj je ta namenjena zgolj za pomoč izbiri lokacije za gradnjo stolpa. S to metodo ustvarimo efekt "povleci in spusti"(ang. "drag and drop").

SelectTowerInCell

Metoda *SelectTowerInCell* nastavi stolp, ki se nahaja na celici trenutnega dotika, kot izbran stolp (objekt *_selectedTower*). Če v tej celici ni stolpa, se vrednost objekta *_selectedTower* pobriše.

BuildTower

V metodi *BuildTower* se zgradi stolp tipa, ki je trenutno shranjen v objektu *_towerToBuild*. Pri tem se najprej preveri, če je celica za gradnjo prosta. Nato se ustvari instanca tipa *_towerToBuild* in se doda v seznam stolpov (*_towers*). Na koncu se igralcu odšteje ustrezno število virov. Takšno delovanje je razvidno iz kode 6.6.

UpdateTowers

Tukaj se sprehodimo skozi seznam vseh stolpov na sceni in jih posodobimo. Po posodobitvi posameznega stolpa se preveri, če mu je morda padla vzdržljivost (*_durability*) na 0. V tem primeru stolp odstranimo iz seznama.

UpdateBullets

V metodi *UpdateBullets* se sprehodimo skozi seznam vseh izstrelkov

na sceni in jih posodobimo. Po posodobitvi posameznega izstrelka se preverita 2 pogoja: prvi je morebiten zadetek sovražnikove enote, drugi pa preveri, če enote morda ni več na sceni. V obeh primerih je potrebno izstrelek odstraniti iz seznama. Če je izstrelek zadel enoto, je potrebno enoti zmanjšati zdravje za odmerek škode izstrelka.

Koda 6.6: Gradnja novega stolpa.

```
Tower newTower = Activator.CreateInstance(_towerToBuild,
    _towerTypes.TowerTextures, _towerTypes.BulletTextures, _towerTypes.PortraitTextures,
    _touchTile, new Vector2(_map.TileSize, _map.TileSize), _bullets) as Tower;

_towers.Add(newTower);
_towerToBuild = null;

// Substract resources.
PlayerResources tempResources = new PlayerResources
{
    Wood = 2
};

switch (newTower.CostMaterial)
{
    case MaterialType.Wood:
        tempResources.Wood += newTower.CostAmount;
        break;
    case MaterialType.Stone:
        tempResources.Stone += newTower.CostAmount;
        break;
    case MaterialType.Iron:
        tempResources.Iron += newTower.CostAmount;
        break;
    case MaterialType.Gold:
        tempResources.Gold += newTower.CostAmount;
        break;
    case MaterialType.Diamond:
        tempResources.Diamond += newTower.CostAmount;
        break;
}

_resources = _resources - tempResources;
```

6.8 Uporabniški vmesnik

Uporabniški vmesnik igre (ang. head up display) sestavlja več razredov: *MainHud*, *MenuHud*, *PlayableHud*, *SwitchHud* in *PopupHud*.

6.8.1 MainHud

Razred *MainHud* se uporablja za prikaz trenutnega stanja igre. Sam razred že prikazuje življenje in vire igralca. Ti podatki so na zaslonu prikazani ves čas. Drugi del uporabniškega vmesnika pa se spreminja s potrebo igralca in je razdeljen na 3 razrede: *InfoHud*, *BuildHud* in *TowerHud*. Za izbiro tega dela uporabniškega vmesnika je na voljo metoda *SetHudType(hudType)*, ki nastavi tip vmesnika za prikaz na vrednost parametra *hudType*.

InfoHud prikazuje podatke trenutne stopnje in njenih enot. To je osnovni prikaz in je izbran ob inicializaciji scene igre.

BuildHud je seznam vseh stolpov. Igralec se lahko dotakne stolpa in ga povleče na ozemlje v prazno celico. Ob dotiku se pokliče metoda *SetTowerToBuild* razreda *Player*. Gradnjo stolpa nato prevzame upravljalec *Player*.

TowerHud prikazuje podatke o stolpu, ki je trenutno izbran. Trenutno izbran stolp pridobimo iz upravljalca *Player*.

6.8.2 MenuHud in SwitchHud

Razred *MenuHud* prikazuje gumb za dostop do glavnega menija in gumb za pavzo. Razred *SwitchHud* pa vsebuje gumb za izbiro vmesnika *InfoHud* in gumb za izbiro vmesnika *BuildHud*.

6.8.3 PlayableHud

Razred *PlayableHud* je namenjen risanju elementov vmesnika na področju ozemlja. V tem razredu se narišejo ploščice, ki prikazujejo zdravje sovražnikovih enot. Poleg ploščic se nariše tudi okvir, ki obkroža trenutno izbran stolp.

6.8.4 PopupHud

Razred *MainHud* se uporablja za prikaz pojavnih oken (ang. popup). Za prikaz takšnega okna pokličemo metodo *ShowPopup(text, offset)*. Pri tem podamo prikazno besedilo kot vrednost parametra *text*, lokacijo okna pa kot vrednost parametra *offset*. Ko želimo pojavno okno odstraniti, pokličemo metodo *HidePopup()*.

Poglavje 7

Sklepne ugotovitve

Skozi razvoj igre Minedefense smo prikazali, da je tehnologija XNA še vedno aktualna. Kljub temu, da se prvotna knjižnica XNA ne razvija več, obstajajo alternative. Originalu se najbolj približa knjižnica MonoGame. Tudi naša igra Minedefense uporablja to knjižnico.

Brez večjih sprememb v programski kodi bi lahko prenesli igro Minedefense tudi na druge mobilne platforme: iOS, PlayStation Mobile in Linux. Spremeniti bi morali samo specifične nastavitve, na katere vpliva ciljni operacijski sistem. Knjižnica omogoča tudi razvoj za stacionarne sisteme: MacOS, Windows, Linux. Pri prenosu na stacionarne sisteme se stvar nekoliko zaplete, saj tam nimamo zaslona na dotik, poleg tega pa je zaslon tudi večji. Prilagoditi bi morali način interakcije igralca z igro ter velikost tekstur.

Knjižnica MonoGame ima tudi svoje pomanjlkjivosti in napake. Največ napak je ravno v delu knjižnice, ki je specifičen za Windows Phone. To je tudi pričakovati, saj je Windows Phone relativno nov sistem, in zato še ni popolnoma podprt. Napaka, ki jo velja na tem mestu omeniti, je napačno branje lokacije dotika na zaslonu. Funkcija, ki vrne pozicijo zadnjega dotika, je vedno vračala točko na neskončni lokaciji. DLL, ki ga MonoGame uporablja za komunikacijo s sistemom Windows Phone je namreč napačno prikazoval meje zaslona. Kot največjo pomanjkljivost velja omeniti odsotnost cevovoda vsebine. Brez cevovoda vsebine, ki bi samodejno pretvoril teksture, zvočne

efekte in pisave v .NET format, je dodajanje vsebine zamudno. Vsebino je potrebno ročno pretvoriti in posneti na primerno lokacijo. Cevovod vsebine bo knjižnica verjetno dobila v eni izmed prihodnjih različic.

7.1 Razvoj iger brez XNA

Kot smo že omenili knjižnica XNA v ozadju uporablja obsežnejšo in zahtevnejšo knjižnico DirectX. Podobno velja za knjižnico MonoGame. Ta bazira na knjižnici OpenGL. Alternativno bi torej lahko uporabili eno izmed naštetih knjižnic. Obe knjižnici sta precej kompleksnejši in bi za uporabo le-teh potrebovali več znanja in časa. Za razvoj preprostih iger, kot je Minedefense, to nikakor ni optimalna izbira.

Igre lahko razvijamo tudi na povsem drugačen način. Zgoraj naštete tehnologije (XNA, DirectX, MonoGame in OpenGL) so zgolj knjižnice za razvoj iger. Vso potrebno fiziko, branje vhodnih podatkov, predvajanje zvočnih efektov, izris grafike in podobno moramo implementirati sami. Potrebno je paziti na vrstni red izvajanja. Tem težavam se lahko izognemo z uporabo pogonov, ki za vse to poskrbijo sam. Razvijalec skrbi samo za igralnost, določa fizikalne lastnosti objektov ter oblikuje teren in objekte. Najpopularnejši pogoni so UDK, Unity in CryEngine.

7.2 Izboljšave igre Minedefense

V igri Minedefense smo implementirali večino funkcionalnosti, ki so bile v načrtih, nekatere pa smo morali iz različnih razlogov izpustiti. Tukaj velja omeniti predvsem algoritem za naključno generiranje ozemlja. Z implementacijo takšnega algoritma bi pridobili na raznolikosti med ponovitvami igranja. Vendar implementacija takšnega algoritma ni enostavna. Lahko se zgodi, da pot ne bi bila zvezna in sovražnik ne bi nikoli dosegel cilja. Ko bi rešili problem zveznosti, bi bilo potrebno paziti, da pot ne bi bila prekratka ali predolga. Nato bi prišlo na vrsto preverjanje terena za gradnjo stolpov. Ta

bi moral biti približno enakovreden med vsemi ponovitvami igranja.

Načrtovali smo tudi izdajo igre na portalu Xbox Marketplace. Za izdajo igre potrebujemo Microsoft Developer Account. Tega računa trenutno nimamo, zato uradno igre ni mogoče izdati. Izdajo igre načrtujemo v eni izmed prihodnjih različic, v kateri bomo dodali tudi naključno generirano ozemlje.

Literatura

[1] (2013) Wikipedia, "Tower defense". Dostopno na:
http://en.wikipedia.org/wiki/Tower_defense

[2] (2013) Wikipedia, "Rampart (video game)". Dostopno na:
http://en.wikipedia.org/wiki/Rampart_(video_game)

[3] (2010) Tower Defense Heaven, "History of tower defense games". Dostopno na:
http://towerdefenseheaven.com/content/history-tower-defense-games

[4] (2013) Element Tower Defense, "Element Tower Defense". Dostopno na:
http://eletd.com/

[5] (2008) Epic War, "Wintermaul TD". Dostopno na:
http://www.epicwar.com/maps/78651/

[6] (2010) Legion TD Official, "Legion TD". Dostopno na:
http://www.legiontd.com/index.php

[7] (2007) Free Web Arcade, "Flash Element TD". Dostopno na:
http://www.freewebarcade.com/game/flash-element-td/

[8] (2013) Armor Games, "Desktop Tower Defense". Dostopno na:
http://armorgames.com/play/1128/desktop-tower-defense-15

[9] (2013) Hidden Path Entertainment, "Defense Grid: The Awakening". Dostopno na:
http://www.hiddenpath.com/games/defense-grid/

[10] (2013) PopCap Games, "Plants vs. Zombies". Dostopno na: http://www.popcap.com/plants-vs-zombies

[11] (2013) Robot Entertainment, "Orcs Must Die!". Dostopno na: http://www.robotentertainment.com/games/orcsmustdie

[12] (2013) Minecraft Wiki, "Minecraft". Dostopno na: http://www.minecraftwiki.net/wiki/Minecraft_Wiki

[13] (2013) Wikipedia, Č Sharp (programming language)". Dostopno na: http://en.wikipedia.org/wiki/C_Sharp_(programming_language)

[14] (2013) Wikipedia, Čommon Language Runtime". Dostopno na: http://en.wikipedia.org/wiki/Common_Language_Runtime

[15] (2013) Wikipedia, "Microsoft Visual Studio". Dostopno na: http://en.wikipedia.org/wiki/Visual_studio

[16] (2013) Wikipedia, "Paint.NET". Dostopno na: http://en.wikipedia.org/wiki/Paint.net

[17] (2013) Wikipedia, "Windows Phone". Dostopno na: http://en.wikipedia.org/wiki/Windows_Phone

[18] (2013) Wikipedia, "Metro (design language)". Dostopno na: http://en.wikipedia.org/wiki/Metro_Design_Language

[19] (2012) Jani Nevalainen, "Windows Phone 8 Kernel Architecture". Dostopno na: http://j4ni.com/blog/?p=107

[20] D. Probert, "Architecture of the Windows Kernel", 2008.

[21] (2006) MSDN Blogs, "What is the XNA framework". Dostopno na: http://blogs.msdn.com/b/xna/archive/2006/08/25/what-is-the-xna-framework.aspx

[22] (2009) Ioannis Panagopoulos, "XNA Game Development (First Steps)". Dostopno na: http://www.progware.org/Blog/post/XNA-Game-Development-(First-Steps).aspx

[23] (2013) Wikipedia, "Microsoft XNA". Dostopno na: http://en.wikipedia.org/wiki/Microsoft_XNA

[24] (2011) XNI - XNA for iOS, "XNI - XNA for iOS". Dostopno na: http://xni.retronator.com/post/2850676376/introduction

[25] (2013) MonoGame, "MonoGame - Write Once, Play Everywhere". Dostopno na: http://monogame.codeplex.com/releases/view/102870

[26] (2013) Gamasutra, "It's official: XNA is dead". Dostopno na: http://www.gamasutra.com/view/news/185894/Its_official_XNA_is_dead.php

[27] S. Russell, P. Norvig, "Artificial Intelligence: A Modern Approach (3rd Edition)", 2009, str. 64–235.