LATVIJAS UNIVERSITĀTE
DATORIKAS FAKULTĀTE

SPĒLES IZSTRĀDE WINDOWS PHONE 7 MOBILAJIEM TELEFONIEM

BAKALAURA DARBS

Autors: **Dmitrijs Ivanovs**
Stud. apl. di07001
Darba vadītājs: Uldis Straujums, Dr. dat.

RĪGA 2011

ANOTĀCIJA

Windows Phone 7 ir jaunākā Microsoft kompānijas mobilā operētājsistēma, kura ātri attīstās. Bakalaura darbs „Spēles izstrāde Windows Phone 7 mobilajiem telefoniem” ir veltīts šīs platformas pētīšanai. Pagaidām operētājsistēma nav plaši izplatīta un tai ir vairāki trūkumi. Darba mērķis ir izpētot šīs platformas iespējas un pieejamās tehnoloģijas, izstrādāt mobilo spēli. Autors pievērš uzmanību spēles izstrādes problēmām un to risināšanas metodēm un iespējām.

Rezultātā ir izstrādāta Windows Phone 7 mobilā spēle, apskatīta operētājsistēmas pamatfunkcionalitāte un atrisinātas vairākas mobilās spēles izstrādes problēmas.

Atslēgvārdi: mobilās platformas, spēles izstrāde, Windows Phone 7, XNA.

ANNOTATION

Windows Phone 7 is the newest mobile operating system from Microsoft. Bachelor's thesis "Windows Phone 7 game development" analyzes this platform. At this point in time the platform is not widely used and has some functionality problems. Work's purpose is to develop mobile game and research platform with available technologies. The author emphasizes the challenges of mobile game development.

As a result, a game for Windows Phone 7 operating system has been developed, the basic functionality of the operating system has been analyzed and main mobile game development related problems have been resolved.

SATURS

IEVADS

Mūsdienās aktīvi attīstās mobilās tehnoloģijas un operētājsistēmas. Mobilās platformas nepārtraukti konkurē sava starpā, cenšoties piedāvāt plašākas iespējas lietotājiem un izstrādātājiem.

Darba mērķis ir iepazīsties ar jaunākās Windows Phone 7 mobilās operētājsistēmas iespējām, izstrādāt šai sistēmai spēli un veikt tās realizācijas izpēti.

Darba sākumā tika definēti sekojoši darba uzdevumi:

- Īsumā iepazīties ar dažādām mobilajām platformām.
- Apskatīt Windows Phone 7 mobilo operētājsistēmu no programmētāja un lietotāja viedokļiem.
- Izpētīt operētājsistēmas pieejamās tehnoloģijas un izvēlēties piemērotāko.
- Iepazīties ar mobilās spēles izstrādes procesu un problēmām.
- Balstoties uz iegūtajām zināšanām, izstrādāt spēli.

Darbs sastāv no 6 nodaļām.

Pirmajā nodaļā ir aprakstītas mūsdienu mobilās operētājsistēmas un dots priekšstats par platformās pieejamajām tehnoloģijām un tendencēm viedtālruņu pasaulē.

Otrajā nodaļā ir detalizēti apskatīta jaunāka *Microsoft* kompānijas Windows Phone 7 mobilā operētājsistēma. Ir aprakstīta pieejamā funkcionalitāte un pievērsta uzmanība gan platformas trūkumiem, gan labumiem.

Trešajā nodaļā apskatītas pieejamās izstrādes tehnoloģijas un dots priekšstats par mobilās spēles izstrādes problēmām un to iespējamajiem risinājumu variantiem.

Ceturtajā nodaļā, balstoties uz iegūtajām zināšanām, ir aprakstīta darba praktiskā daļa - mobilās spēles realizācija. Šajā nodaļā ir aprakstīta spēles izstrādes gaita, sākot ar spēles ideju un beidzot ar spēles izplatīšanu, kā arī ir aprakstītas sastopamās problēmas un to risinājumi.

Piektā nodaļa ir rezultātu un secinājumu nodaļa, kurā uzskaitīti iegūtie rezultāti un izdarīti secinājumi par paveikto darbu un atrisinātām problēmām.

Sestajā nodaļā ir saraksts ar informācijas avotiem.

Septītajā nodaļā ir pievienoti pielikumi – spēles pirmkoda galvenās daļas un vienībtestu fragments.

Astotajā nodaļā ir pievienota dokumentārā lapa.

1. MOBILĀS OPERĒTĀJSISTĒMAS

Mūsdienās aktīvi attīstās mobilās tehnoloģijas. Mobilie telefoni pārvēršas par tikpat daudzfunkcionālām ierīcēm kā datori. Šodien mobilie telefoni tiek lietoti arvien mazāk un tiek aizvietoti ar viedtālruņiem. Viedtālrunis ir ierīce ar visām mobilajam telefonam raksturīgām funkcijas, kas papildināta ar datoram līdzīgu funkcionalitāti [4].

Viedtālrunis nodrošina ļoti plašas funkcijas. Nozīmīgākās no tām ir:

- zvanu pieņemšana un zvanīšana,
- SMS[1] un MMS[2] īsziņu sūtīšana un saņemšana,
- kalendārs ar plānotāju,
- kontaktu grāmatiņa,
- audio un video atskaņošana,
- foto un video uzņemšana,
- pieeja globālajam tīklam caur pārlūkprogrammām,
- e-pasta klients ar e-pasta sūtīšanas un saņemšanas iespējām,
- balss atpazīšana,
- iespēja ērti instalēt citas lietojumprogrammas,
- GPS[3] navigācija,
- akselerometra[4] izmantošana,
- Bluetooth[5] izmantošana.

Tik plaša funkcionalitāte ir pieejama, pateicoties telefona aparatūras un operētājsistēmas uzlabošanai vai arī jaunas operētājsistēmas izveidošanai.

---

[1] Short Message Service (SMS) – pakalpojums, kas ļauj lietotājam sūtīt un saņemt īsus ziņojumus.

[2]Multimedia Messaging Service (MMS) – pakalpojums, kas ļauj sūtīt un saņemt ziņojumus ar multivides saturu.

[3]Global Positioning System (GPS) – pavadoņu navigācijas sistēma objekta atrašanas vietas noteikšanai.

[4]Akselerometrs - ierīce, kas mēra paātrinājumu. Viedtālruņos tiek pielietota virziena un stāvokļa noteikšanai.

[5]Bluetooth – maza darbības rādiusa bezvadu datortīklu standarts. Šī tehnoloģija piedāvā iespēju apmainīties ar informāciju, savienojoties ar dažādām ierīcēm.

1.1. Mobilās operētājsistēmas

Katrā mobilajā telefonā ir operētājsistēma, kura kontrolē ierīci, līdzīgā veidā kā operētājsistēma kontrolē datoru, tomēr mobilā operētājsistēma ir vienkāršāka. Vairākas kompānijas nodarbojas ar mobilas operētājsistēmas izstrādi un cenšas izveidot labāku, ātrāku un funkcionālāku operētājsistēmu. Lielākie tirgus dalībnieki ir: *Apple*, *Google*, *Microsoft*, *Nokia*, *Samsung*, *Research In Motion (RIM)* un *HP*. Daži no tiem veido arī savas ierīces, bet daži – tikai piedāvā savas operētājsistēmas kompānijām, kuri ražo viedtālruņus.

Lielāko soli uz priekšu izdarīja kompānija *Apple* 2007 gadā, izveidojot unikālu telefonu – *iPhone* ar operētājsistēmu iOS. Gan ierīce, gan operētājsistēma ir izstrādāti kompānijā *Apple*. Šīs telefons apvienoja sevī satriecošu skārienjūtīgu ekrānu, kurš spēj atpazīt vairākus vienlaicīgus pieskārienus un lietotājam draudzīgu operētājsistēmu, kura fokusējas uz skārienjūtīgo saskarni. Nedaudz vēlāk nākamo soli izdarīja *Google*, izstrādājot Android operētājsistēmu, kura kļuva par galveno konkurenti iOS platformai, un tikai 2010. gadā *Microsoft*, cenšoties panākt konkurentus, izveidoja Windows Phone 7 mobilo operētājsistēmu [29].

Viedtālruņu tirgus ir ļoti dinamisks. Detalizētākā informācija par dažādu platformu izplatību [5] ir apkopotā tabulā 1.1.

1.1. tabula

Mobilo operētājsistēmu izplatība pasaulē

Gads \ OS	Symbian	iOS	BlackBerry OS	Android	Others
2008	42 %	33 %	-	-	25 %
2009	36 %	34 %	8 %	3 %	19 %
2010	32 %	26 %	16 %	9 %	17 %
2011	31 %	25 %	15 %	16 %	13 %

Viena no svarīgākajām operētājsistēmas sastāvdaļām ir lietojumprogrammas, kuras paplašina un attīsta platformu un kuras raksta programmētāji no visas pasaules. Tāpēc ir svarīgi piedāvāt visiem ieinteresētiem programmētājiem draudzīgu vidi jaunas platformas apgūšanai.

Nozīmīgākie faktori programmētājiem ir sekojoši:

- platformas izplatība,
- platformas attīstība un nākotnes perspektīvas,
- platformas funkcionalitāte,
- programmētāja dokumentācija,
- piemēru esamība,
- ērti un pieejami izstrādes rīki ar telefona emulatoru,
- iespēja izplatīt izstrādāto programmatūru.

1.1.1. Windows Phone

Pašlaik eksistē divas *Microsoft* mobilās operētājsistēmas: veca Windows Mobile un jauna Windows Phone 7. Vecā operētājsistēma bija izveidota 2000. gadā un, tai novecojot, tā pakāpeniski tika aizstāta ar 2010. gadā izveidoto, pilnīgi jaunu Windows Phone 7 operētājsistēmu, kura neatbalsta vecās versijas lietojumprogrammas. Jaunā Windows Phone 7 atšķiras no citām platformām ar unikālu lietotāja saskarni.

Windows Phone 7 programmatūras izstrādei iespējams pielietot XNA vai Silverlight rīkkopu. XNA ietvars[1] ir bāzēts uz .NET Compact Framework 2.0 priekš Xbox 360 spēles konsoles un ietver sevī bibliotēkas spēļu izstrādei, savukārt Silverlight tehnoloģija ir līdzīga Adobe Flash tehnoloģijai, kura paplašina Interneta vietnes ar grafiku un animāciju. Tehnoloģijas pielietošanai nepieciešama Visual Studio izstrādes vide un C# programmēšanas valodas zināšanas (Visual Studio 2010 versijā XNA tehnoloģijai ir pieejama arī Visual Basic valoda).

Programmatūras izplatīšanai šobrīd vienīgā iespēja ir Windows Phone Marketplace veikals. Programmētājiem, piedaloties tajā, nepieciešams katru gadu maksāt 99$ (studentiem pieeja veikalam ir bezmaksas) un 30% ienākumu no programmatūras pārdošanas *Microsoft* patur.

[1] Ietvars (*Framework*) - platforma ar definētu un strukturētu vidi, ar kuras palīdzību iespējams izstrādāt programmatūru.

1.1.2. Symbian OS

Symbian ir atklātā pirmkoda operētājsistēma, kura ir izstrādāta speciāli priekš viedtālruņiem. Ar tās izstrādi un uzturēšanu pašlaik nodarbojas kompānija *Nokia*. Tā apvieno sevī vairākas platformas: S60, UQI un MOAP(S) lietotāja saskarni. Operētājsistēma ir izstrādāta *Symbian Ltd*. kompānijā, kuru 2008. gadā nopirka Nokia, uzreiz izveidojot bezpeļņas organizāciju *Symbian Foundation*, kura nodarbojas ar kopēja standarta izveidi un Symbian pirmkoda izplatīšanu [6]. Šobrīd tā ir vispopulārākā mobilā operētājsistēma pasaulē, kuras izplātīšanu tuvākajos gados, pēc jauna līguma starp Nokia un Miscrosoft, plānots samazināt [7].

Sākot ar 2010. gadu Symbian pārgāja uz C++ standartu ar Qt programmatūras izstrādātāja rīkkopu, kura atbalsta vecas un jaunas Symbian versijas.

Symbian atbalsta vairākas alternatīvas lietojumprogrammu izstrādes tehnoloģijas [6]:

- Python
- Flash Lite
- Java ME
- Ruby
- .NET

Symbian lietojumprogramma ir sapakota SIS failā, kuru var brīvi instalēt, pieslēdzot telefonu datoram, caur Bluetooth, vai arī pa tiešo uzinstalējot uz atmiņas karti. Nokia mobilajiem telefoniem programmas instalēšanai ir pieejams arī *Nokia* serviss Ovi. Izstrādātājiem reģistrācija tajā maksa 1€ un kompānija patur 30% no peļņas.

1.1.3. iOS

Kompānijas *Apple* operētājsistēma, kas izstrādāta 2007. gadā priekš iPhone viedtālruņiem, pašlaik tiek izmantota arī iPod touch, iPad un Apple TV ierīcēs. iOS ir Unix līdzīga sistēma, kas atvasināta no Mac OS X datora operētājsistēmas [8].

Lietojumprogrammas, kas paredzētas šai operētājsistēmai, nepieciešams rakstīt un kompilēt speciāli priekš iOS platformas ar ARM arhitektūru. Izstrāde iespējama vienīgā programmēšanas valodā – Objective-C (*Apple* kompānijas kompilējamā objektorientētā programmēšanas valoda, bāzēta uz C valodas un Smaltalk paradigmām), kā arī visiem iOS un

Mac programmas dalībniekiem ir pieejama izstrādes vide Xcode (tikai priekš Mac OS operētājsistēmas) ar iebūvētu iPhone emulatoru.

Vienīgā oficiāla programmatūras izplatīšanas vieta ir App Store veikals, kurā iespējams publicēt savas programmas, izejot to pārbaudi un nosakot cenu. App Store ir lielākais mobilās programmatūras izplatīšanas veikals pasaulē ar aptuveni 300 000 programmām. Lai piekļūtu veikalam, programmētājam nepieciešams reizi gadā maksāt 99$, un 30% no peļņas patur *Apple*.

1.1.4. Android

Android ir *Google* kompānijas atklātā pirmkoda operētājsistēma priekš viedtālruņiem. Tā ir bāzēta uz modificēta Linux kodola. Pirmais telefons ar Android sistēmu bija pieejams 2008. gadā. Android platformu plaši izmanto lielākie viedtālruņu ražotāji: Samsung, HTC, LG un citi. Kopš tā laika sistēma aktīvi attīstās un pašlaik ir izplatītākā platforma pasaulē.

Primāra izstrādes programmēšanas valoda ir Java, bet lietotāja saskarnes definēšanai tiek pielietota XML valoda. Programmatūras izstrādātāja rīkkopa Android SDK satur plašus izstrādes rīkus un informāciju -atkļūdotājs, bibliotēkas, emulators, dokumentācija, piemēri un pamācības. Standarta izstrādes vide Android platformai ir Eclipse ar ADT (Android Development Toolkit) spraudni, kura ir pieejama uz vairākām platformām (Linux, Mac OS X, Windows).

Programmatūras izplatīšanai ir pieejamas vairākas vietas - Android Market ar aptuveni 200 000 programmām, dalība kurā programmētājam izmaksā 25$ (samaksājot vienreiz) un *Google* patur 30% no peļņas, kā arī nesen atvērtais Amazon kompānijas Appstore for Android.

Kā alternatīva Android viedtālruņos ir pieejams Flash, tātad lietojumprogrammas var izstrādāt arī pielietojot Flash tehnoloģijas.

1.1.5. Citas mobilās platformas

- **BlackBerry OS**

RIM kompānijas operētājsistēma priekš BlackBerry viedtālruņiem, kas plaši izplatīta Ziemeļamerikā. Programmēšanas valoda ir Java ar plašām bibliotēkām. Lietojumprogrammu izplatīšana notiek caur Blackberry App World veikalu ar aptuveni 15 000 lietojumprogrammām.

- **Bada**

Samsung Electronics kompānijas mobilā operētājsistēma, kuras mērķis ir aizstāt veco Samsung operētājsistēmu. Tā ir izstrādāta 2010. gadā. Izstrādei priekš šīs operētājsistēmas nepieciešams zināt C++. Izstrādes vide, tāpat kā Android platformai, ir Eclipse ar Bada SDK rīkkopu, kura satur visu nepieciešamo izstrādei – sakot no atkļūdotāja līdz lietotāja saskarnes veidotājam un emulatoram, kurā var palaist izstrādāto programmatūru. Lietojumprogrammas izplatīšanai ir pieejams Samsung Apps veikals.

- **MeeGo**

Linux bāzētā atklātā koda operētājsistēma priekš viedtālruņiem un planšetdatoriem. Platformu aktīvi attīsta no 2010. gada *Intel*, *AMD*, *Nokia* un *Novell* kompānijas un šogad tirgū jāparādās pirmajiem telefoniem ar šo sistēmu. Izstrādei ir pieejams MeeGo SDK, programmēšanai tiek pielietota C++, izmantojot Qt ietvari un Qt Creator vidi.

- **WebOS**

Agrāk zināma kā Palm OS, bet pēc Palm nopirkšanas ar HP kompāniju 2009. gadā tika pārsaukta par WebOS. Šīmobilā operētājsistēmaizmanto Linux kodolu. Izstrādei var pielietot HTML, CSS un JavaScript vai arī C++.

1.2. Kopsavilkums un secinājumi

Nodaļā apskatīto operētājsistēmu kopsavilkums [6] ir apkopots tabulā 1.2. No aplūkotas informācijas var secināt, ka mobilo operētājsistēmu skaits ir liels un programmētājam ir nepieciešams zināt vairākas tehnoloģijas. Gadījumā, ja ir nepieciešams uzrakstīt vienu lietojumprogrammu vai spēli vairākām platformām, būs nepieciešams laiks, lai apgūtu tās visas, kā arī būs nepieciešams naudas ieguldījums, lai spētu uzrakstīt un izplatīt uzrakstīto programmu, jo izplatīšana populārākajās platformās notiek caur vienīgu veikalu, kurā nepieciešams maksāt par dalību tajā.

1.2. tabula

Mobilo operētājsistēmu salīdzināšana

	Kompānija	**Valoda**	**Licence**	**SDK platforma**	**OS saime**	**Izplatīšana**	**Trešās puses**[1]
Windows Phone	Microsoft	C#	Maksas	Windows	Windows CE 7	Windows Marketplace	-
Symbian OS	Symbian Foundation	C++, Java	Atvērta	Vairākas	S60	Nokia Ovi store	+
iOS	Apple	ObjectiveC	Maksas	Mac OS X	Mac OS X	App Store	-
Android	Google	Java	Atvērta	Vairākas	Linux	Android Market	+
BlackBerry OS	RIM	Java	Maksas	Linux	-	App World	+
Bada	Samsung	C++	Maksas	Windows	Linux	Samsung Apps	-
MeeGo	Linux Foundation	C++	Atvērta	Linux	Linux	Nav[2]	+
WebOS	HP	C++, JavaScript	Daļēji atvērta	Vairākas	Linux	App Catalog	-

Apskatot vairākas populārākās platformas, var secināt, ka tās ir līdzīgas gan pēc lietotāja saskarnes, gan pēc satura. Jaunāka *Microsoft* kompānijas mobilā operētājsistēma Windows Phone 7 ar tās lietotāja saskarnes ideju, kura tiks aprakstītā darbā, ir izdarījusi nākamo soli uz priekšu mobilās platformas tirgū. Pagaidām platformas izplatīšanas un popularitātes procents ir ļoti niecīgs, tādēļ kā ir sistēma pieejama tikai dažās valodās un operētājsistēmas funkcionalitāte nav tikpat augstā līmenī kā parējās platformās. Neskatoties uz trūkumiem, autors uzskata, ka tā ir viena no perspektīvākajām jaunajām platformām, kura ātri attīstīsies un sasniegs pasaules tirgus virsotni. Nesen parakstītais līgums starp *Nokia* un *Microsoft* par sadarbību un pāreju no vecās operētājsistēmas uz Windows Phone 7 operētājsistēmu sekmēs platformas izplatīšanu pasaulē.

[1]Iespēja instalēt trešās puses lietojumprogrammas neizmantojot oficiālo izplatīšanas veikalu

[2]Tuvākajā laika lietojumprogrammas ir plānots izplatīt Ovi Store un Intel AppUp veikalos

2. WINDOWS PHONE 7 APSKATS

Windows Phone 7 ir jauna *Microsoft* kompānijas Windows CE saimes mobilā operētājsistēma, kas izstrādāta 2010. gadā. *Microsoft* nenodarbojas ar ierīču izstrādi un tikai piedāvā jaunu platformu ierīču ražotājiem. Sākotnēji *Microsoft* ir definējuši minimālās prasības pret telefona aparatūru, un tās ir sekojošas:

- skārienjutīgais ekrāns ar 480x800 izšķirtspēju,
- 1 GHz procesors un 256 MB operatīva atmiņa,
- iebūvētās iekārtas: akselerometrs, gaismas sensors, fotokamera un radio,
- 6 pogas.

Pašlaik pasaulē ir pieejami 10 viedtālruni ar šo platformu.

2.1. Galvenās raksturiezīmes

- **Lietotāja saskarne**

Windows Phone 7 platformai ir pilnīgi jauna lietotāja saskarne – Metro, kura atšķir šo operētājsistēmu no citām. Metro saskarnes pamatā ir poligrāfijas dizains ar lieliem fontiem, kas piesaista lietotāja uzmanību. *Microsoft* dizaineru komanda ir izveidojusi izaicinošo dizainu, kurā tiek realizēti drosmīgi risinājumi – lietotājs, balstoties uz redzamajām teksta vai bildes daļiņām, saprot, ka ir iespējams apskatīt informāciju blakus esošajās lapaspusēs (2.1. attēls).

Metro dizaina pamatprincipi [16] ir:

- **Tipogrāfija**

  Burti ir ne tikai skaisti, bet arī funkcionāli – teksta izmēra bilance vizuāli veido hierarhiju, bet pareizi izvēlēts izkārtojums palīdz aptvert vairāk satura un viegli to atrast.

- **Kustība**

  Kustība ir tas, kas „atdzīvina" saskarni. Izveidotā vienota animācijas apakškopa padara sistēmas izmantojamību ērtāku un saprotamāku.

- **Saturs**

  Viens no unikālajiem principiem Metro saskarnē. Saskarnē izslēdzot visus papildus dekorācijas elementus, uzmanība fokusējas uz satura. Tas ir īpaši svarīgi uz mazāka izmēra ekrāniem un žestu balstītā saskarnē.

2.1. att. **Windows Phone 7 Metro lietotāja saskarne**

- **Pamatvirsma**

Windows Phone 7 pamatvirsma ir realizēta, izmantojot interaktīvās „flīzes", kuras aizvieto ierastās ikonas. „Flīzes" atspoguļo dažāda tipa papildinformāciju – tā varētu būt neatbildēto zvanu skaits, jaunas fotogrāfijas utt. (2.2. attēls).

2.2. att. **Windows Phone 7 pamatvirsma**

- **Ietvari**

Ietvari (hubs) apvieno sevī vienāda tipa informāciju. Tie ļauj ērti vienā vietā apskatīt visu interesējošo informāciju. Kontaktu grāmatiņa ir apvienota ar Facebook sociālo tīklu, spēles ir apvienotas ar Xbox Live servisu, mūzika un video ir apvienoti ar Zune mūzikas veikalu, bet dažādu tipu dokumenti ir apvienoti vienā ietvarā, kur iespējams apskatīt esošus dokumentus un pat rediģēt tos, kā arī veidot jaunus. Informācija apvienota ar populārākajiem *Microsoft* servisiem, kas padara telefona lietošanu daudz ērtāku.

- **Pārlūkprogramma**

Operētājsistēmā ir Internet Explorer Mobile pārlūkprogramma, kura atspoguļo tīmekļa lapas līdzīgi IE8 pārlūkprogrammai. Pārlūkprogrammā ir iespējams atvērt līdz 6 lapām (t.i. 6 cilnēm) vienlaicīgi.

- **Sinhronizācijas iespējas**

Sinhronizācija ar datoru tiek nodrošināta ar Zune programmas palīdzību. Zune programma nodrošina arī operētājsistēmas atjaunošanu.

- **Windows Phone Marketplace veikals**

Vienīgā oficiāla vieta, kur ir apkopotas visas Windows Phone 7 lietojumprogrammas un spēles. Izstrādātājs, reģistrējoties tajā, iegūst iespēju augšupielādēt savas programmas veikalā.

2.2. Funkcionalitātes trūkumi

Tā kā operētājsistēma ir jauna, tā nav ideāla un tajā ir vairāki trūkumi:

- **Valodas** – pašlaik operētājsistēma ir pārtulkota tikai dažās valodās.
- **Vairākuzdevumu režīms** – nav iespējams palaist vairākas programmas vienlaicīgi.
- **Interneta koplietošana caur tālruni**.
- **Lietojumprogrammu daudzums** – platforma ir jauna, tāpēc lietojumprogrammu skaits nav liels.
- **Pielāgošana –** atšķirībā no citām platformām, mainīt sistēmas izskatu var tikai minimāli, izvēloties krāsas un mainot vietām pamatvirsmas elementus.

MIX11 konferencēs ietvaros tika paziņotie sistēmas nākotnes plāni, kuri ietver sevī vairākus uzlabojumus, interesantākie no tiem: vairākuzdevumu režīma realizācija, pārlūkprogramma IE9 ar HTML5 un aparatūras paātrinājumu un iespēja realizēt pamatvirsmas interaktīvas „flīzes" katram programmētājam [30].

2.3. Iespējas izstrādātājiem

Oficiāli lietojumprogrammas izstrādei ir pieejama vienīgā vide – Visual Studio, bet eksistē arī neoficiāla alternatīva – Mono.

2.3.1. Visual Studio

Visual Studio izstrādes vide priekš Windows Phone 7 operētājsistēmas programmēšanas ietver sevī Windows Phone Developer Tools rīkkopu, kura sastāv no vairākiem rīkiem:

- **Windows Phone Emulator**

Mobilas platformas emulators (2.3. attēls)ļauj palaist izstrādāto lietojumprogrammu vai spēli testēšanai.

*2.3. att.***Windows Phone 7 emulators**

Emulators tiek pielietots programmatūras testēšanai un atkļūdošanai, bet emulatoram ir daži trūkumi: tas spēj strādāt ar vairākiem pieskārieniem, bet, lai to izdarītu, ir nepieciešams instalēt speciālu programmatūru, kura ļauj manipulēt ar vairākām pelēm, vai arī nopirkt skārienjūtīgo monitoru. Otrais emulatora trūkums – tas nespēj emulēt akselerometra darbību. Lai to simulētu, var izmantot citu lietojumprogrammu, kura strādā ar videokameras palīdzību. Abi trūkumi nav kritiski, jo uzrakstīto programmatūru iespējams testēt uz reālām ierīcēm un šie trūkumi tiks izlaboti nākamajās versijās.

- **Silverlight**

Silverlight priekš Windows Phone platformas ir izstrādāts uz Silverlight 3 versijas bāzes. Silverlight tehnoloģija ir līdzīga Adobe Flash tehnoloģijai, kura paplašina tīmekļa vietnes un lietojumprogrammas ar multividi. Silverlight tehnoloģija apvieno iezīmēšanas valodu saskarnes definēšanai un programmēšanas valodu. Iezīmēšanai tiek pielietota Extensible Application Markup Language (XAML) – paplašināma lietojumprogrammas iezīmēšanas valoda, kas ļauj definēt elementu izkārtojumu lietotāja saskarnē, bet programmēšanai pielieto C# valodu.

- **XNA Game Studio 4.0**

Rīkkopa, ar kuras palīdzību ir ērtāk programmēt un izstrādāt spēles. XNA ietvars ir bāzēts uz .NET Compact Framework 2.0 ietvara priekš Xbox 360 spēles konsoles un ietver sevī bibliotēkas, kas speciāli paredzētas spēļu izstrādei. Šī rīkkopa ļauj rakstīt spēles, kuras ir iespējams palaist gan uz Xbox konsoles, gan uz Windows Phone 7 telefoniem, gan uz visām Windows operētājsistēmām ar XNA ietvara atbalstu.

- **Microsoft Expression Blend for Windows Phone**

Grafiska vide lietotāja saskarnes veidošanai. Pielietojot šo rīku, ir iespējams izveidot telefona lietojumprogrammas lietotāja saskarni, nerakstot kodu.

2.3.2. MONO

MONO ir atklātā pirmkoda projekts, kura mērķis ir izstrādāt ar .NET saderīgu rīkkopu un ļaut palaist Microsoft .NET programmatūru uz vairākām platformām [21].

Priekš XNA ietvara eksistē vairāki projekti, to starpā MonoGame un arī MonoXNA, kura mērķis ir XNA ietvara realizācija vairākām platformām un nodrošināt labāku XNA spēles pārnesamību uz citam platformām – iOS, Android un Mac OS X. [22, 23]. Šie projekti ir dažādās realizācijas stadijās un tiem trūkst dokumentācijas. Diemžēl jauniem programmētājiem priekš pilnvērtīgas un ātras izstrādes šie projekti vēl nav pielietojami.

3. MOBILĀS SPĒLES

Mobilā spēle ir video spēle, kuru ir iespējams palaist uz viedtālruņiem vai portatīvajiem multivides atskaņotājiem (spēles uz portatīvajām konsolēm netiek sauktas par mobilām spēlēm). Pirmās mobilās spēles parādījās kopā ar pirmajiem mobilajiem telefoniem – tiem telefoniem bija mazi melnbalti ekrāni un skaņas iespējas ierobežojās ar dažāda toņa „trokšņiem". Populārākais piemērs ir čūskas spēle. Tajās dienās spēles salīdzinoši nebija populāras ierīču tehniskās ierobežotības dēļ. Mūsdienās viedtālruņi ir kļuvuši neticami ātri un, pateicoties spilgtiem un krāsainiem lieliem skārienjūtīgiem ekrāniem, mobilās spēles ir sasniegušas augstāko līmeni - vairs nav nepieciešams sēdēt mājās pie datora, spēlēt spēlēs iespējams jebkurā vietā. Viedtālrunis paplašina spēļu iespējas un padara tās vēl interesantākas.

Mobilās spēles atšķiras no parastām video spēlēm sekojošos punktos:

- **Ierīces mobilitāte**

  Viedtālrunis ir portatīva mobilā ierīce, kura ir pieejama visur, tātad spēlēt uzinstalētās spēles iespējams jebkurā vietā.

- **Grafika**

  Datoros ir jaudīgākie video paātrinātāji un vairākkodolu procesori, kuri atļauj uztaisīt ļoti detalizētas spēles. Pēc detalizācijas mobilās spēles stipri no datora video spēlēm ierīces aparatūras dēļ, tāpēc mobilās spēles cenšas piesaistīt uzmanību ar interaktīvu saskarni un pievilcīgām multiplikācijām.

- **Spēles vadība**

  Mobilajās ierīcēs nav kursorsviras, tāpēc spēlēt ir sarežģītāk. Tomēr ir interesantas papildiespējas – skārienjūtīgais ekrāns un akselerometrs ar kuras palīdzību iespējams paplašināt spēles vadību. Piemērotas spēles vadības realizācija ir nopietnāka spēles izstrādes problēma.

- **Ekrāns**

  Viedtālruņiem ir mazi ekrāni – no 2.5" līdz 4.3" (6.35 cm – 10.92 cm), tāpēc ir grūtāk izvietot vairākus objektus vienā ekrānā.

- **Vairākspēlētāju režīms**

  Realizēt vienlaicīgu vairākspēlētāju režīmu pie vienas mazas ierīces bez kursorsviras ir problemātiski, tāpēc jādomā par citām metodēm – Interneta vai Bluetooth tehnoloģijas pielietošanu spēlētāju apvienošanai spēles pasaulē, vai arī spēlēšanas pēc kārtas realizēšanu.

- **Skaņa**

  Neliela izmēra dēļ viedtālrunis nevar atskaņot skaņas tikpat labi kā personāls dators. Lietotājs varētu pieslēgt skaļruņus vai austiņas, lai palielinātu atskaņošanas kvalitāti.

Lietotāji pērk mobilās spēles un lielākas kompānijas piedāvā savus veikalus, kur to ir iespējams izdarīt. Ienākumi no mobilajām spēlēm pasaulē 2009. gadā ir pacēlušies uz neticamu augstumu - 4.7$ miljardi, un joprojām palielinās [10].

Mobilās spēles, tapāt kā parastās video spēles, dalās pēc žanriem. Žanru skaits ir ļoti liels, populārākie no tiem ir atjautības (puzzle) spēles, darbības (action) spēles, arkādes (arcade) spēles, stratēģijas un izglītības spēles [11]. Mobilās spēles viedtālruņos atjaunoja vecas 2D (divdimensionālas) spēles, kuras prasa mazāk resursu un ir ātrāk realizējamas, kā arī neprasa sarežģītu spēles vadību.

3.1. Izstrādes ietvara izvēle Windows Phone 7 platformai

Pirmā problēma ar kuru sastopas izstrādātājs ir piemērotākas tehnoloģijas izvēle priekš Windows Phone 7 operētājsistēmas. Pašlaik ir pieejami divi ietvari - XNA un Silverlight, kuras pagaidām nevar apvienot vienā lietotnē (nākotnē plānots piedāvāt iespēju izmantot divas tehnoloģijas vienā lietojumprogrammā vai spēlē).

Tehnoloģiju salīdzināšana vairākos aspektos ir pieejama tabulā 3.1. [12, 13].

3.1. tabula

XNA un Silverlight salīdzināšana

	Silverlight	**XNA**
Nolūks	Uz XAML bāzēts programmatūras ietvars priekš ātras lietojumprogrammu izstrādes	Augstas veiktspējas spēļu ietvars priekš 2D un 3D spēļu izstrādes
Saskarnes elementi	Izmanto platformas standarta lietotāja saskarnes elementus	Nepieciešams pašiem realizēt vajadzīgos elementus
Video	Iespējams atspoguļot video pat uz nelielas ekrāna daļas	Pieejams pilnekrāna režīms
HTML (pārlūkprogramma)	Iespējams izmantot pārlūkprogrammas elementu	Nav pieejams
Spēles cikls	Jārealizē izstrādātājam	Standarta mehānisms
3D grafika (trīsdimensionāla**)**	Ierobežots (iespējams simulēt 3D)	Atbalsta 3D
Grafiskā aparatūras paātrināšana	Nav pieejama	Ir pieejama

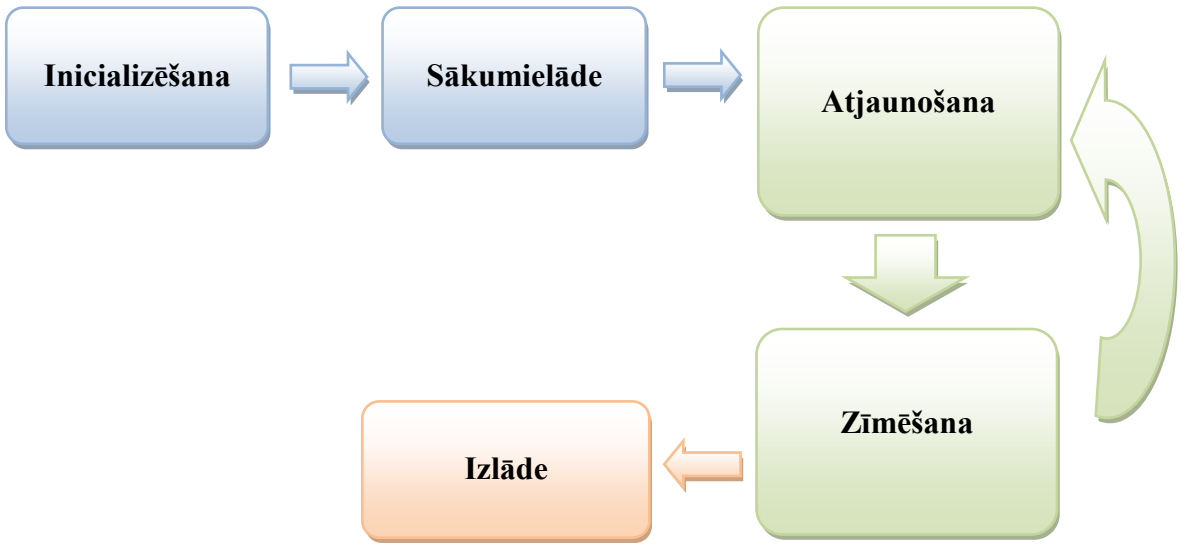
No aplūkotiem datiem viegli secināt, kad un kuru ietvaru pielietot, izstrādājot Windows Phone 7 spēli vai lietojumprogrammu. Silverlight ietvars domāts lietojumprogrammas izstrādei ar standarta platformas elementu izmantošanu ar dažādam multivides papildinājumiem. Pielietot Silverlight 2D spēles izstrādei ir iespējams, bet šīs ietvars nav optimizēts darbam ar lielu grafisko objektu skaitu un 3D grafiku.

Tieši XNA tehnoloģija ir nepieciešama spēles izstrādei priekš Windows Phone 7 platformas. XNA ļauj izmantot aparatūras paātrināšanu grafikai, kā arī ietvars ļauj ērti manipulēt ar dažādiem grafiskajiem objektiem – modeļiem, faktūrām, reljefiem, animācijām, gariņiem un citiem objektiem. XNA tehnoloģijas pielietošanai ir nepieciešams zināt C# programmēšanas valodu.

3.2. XNA ietvars

XNA ietvars ir izveidots tieši spēles izstrādei. Spēlē darbības notiek visu laiku, un programma pārbauda, kad nepieciešams, stāvokli vai notikumus, un reaģē uz to. Spēle ir cikls, kurš notiek visu laiku, kamēr lietotājs neizdomās to pārtraukt. XNA pamatā ir šis cikls (3.1. attēls). Cikls sastāv no vairākām fāzēm [14]:

1. Inicializēšanas fāzē notiek objektu inicializēšana.
2. Sākumielādes fāzē notiek satura ielādēšana. Tie ir grafiskie objekti, skaņas, fonti un citi liela izmēra objekti. Parasti šī fāze ir lēnāka, jo tiek ielādēts liels objektus skaits.
3. Cikls sastāv no divām darbībām:
   - Atjaunošana – tiek atjaunoti nepieciešamie parametri objektiem, kā arī šeit iespējams uztaisīt pārbaudi uz lietotāja veikto darbību (piemēram, pirksta pieskārienu) un tās apstrādi.
   - Zīmēšana – notiek atjaunoto objektu zīmēšana. Ātrums, ar kuru notiek cikla atkārtošana, iespējams definēt programmā. Vērtība pēc noklusējuma ir 1/60 sekundes daļa vienam cikla atkārtojumam. Cikls varētu atkārtoties lēnāk gadījumā, ja notiek aiztures apstrādē un lietotājs varētu ieraudzīt spēles aiztures. Tāpēc izstrādātājiem, cik vien tas ir iespējams, jācenšas samazināt programmas sarežģītību un optimizēt ciklā esošos algoritmus.
4. Izlādes fāzē notiek ielādēto objektu atbrīvošana.



3.1. att. **XNA spēles cikls**

XNA ietvars arī tiek izmantots Xbox video konsoles spēles izstrādei – realizēto spēli iespējams ātri pārveidot no Xbox konsoles uz Windows Phone 7 un otrādi. XNA ietvarā eksistē vairākas iespējas spēles izstrādātājiem, kuras nāk pēc noklusējuma un palīdz spēles izstrādē - darbs ar dažāda tipa ģeometriskām figūrām, vektoriem un tekstūrām.

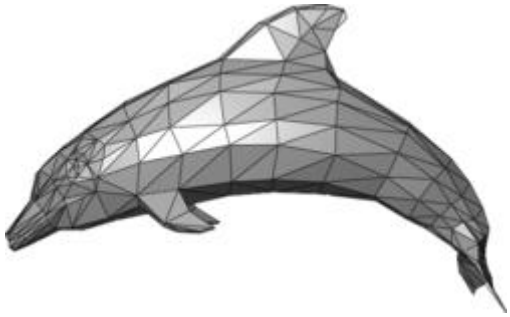
3.3. Grafiskie objekti un animācija

Katra spēle satur daudzus grafiskos objektus. Izstrādātājs, manipulējot ar tiem, atdzīvina tos un izveido spēles loģiku. XNA ietvars ļauj manipulēt ar daudziem grafiskiem formātiem: .bmp, .dds, .dib, .hdr, .jpg, .pfm, .png, .ppm un .tga. Spēles paplašināšanai nepieciešams realizēt dažāda veida animāciju. Animācija padara spēli spilgtāku, dinamiskāku un pievilcīgāku.

2D animācijai, kuru nav iespējams viegli paveikt ar attēla transformācijas palīdzību (mainot objekta izmērus, rotējot) vai ar vairāku grafisko objektu kombināciju un transformāciju, parasti tiek pielietota gariņu tehnoloģija. Gariņš (*Sprite*) – 2D attēls, kurš satur vienu vai vairākus speciāli sakārtotus objektus, kuri nedaudz atšķiras savā starpā (3.2. attēls). Rādot šāda attēla noteiktu daļu pēc noteiktas loģikas (definējot nepieciešamās daļas izmēru un kadra pārejas ātrumu), ir iespējams panākt animācijas efektu. Šī tehnoloģija ir līdzīga metodei, kuru pielieto multiplikatori – zīmējot katru animācijas kadru atsevišķi un pēc tam ātri mainot uzzīmēto, notiek objekta animācija.

3.2. att. **Gariņu piemēri**

3D animācija ir sarežģītāka lieta, kura sastāv no vairākiem etapiem. Sākumā notiek objekta zīmēšana no dažādiem leņķiem, pēc tam tiek veidots 3D modelis. Modelis tiek sadalīts daudzstūra režģī (3.3. attēls), ar kura palīdzību iespējams ātrāk apstrādāt šo modeli un vieglāk uzklāt modelim virsū noteikto faktūru (faktūrkartēšana). Tālāk modelis tiek sadalīts sīkāk – tiek definēti punkti, kur un kā modelis varēs transformēties (piemērs, cilvēka modelī šie punkti būtu skeleta locītavas). Beidzot atliek tikai, pareizi manipulējot ar izveidoto modeli, izveidot animāciju [15].



3.3. att. **3D modeļa trīsstūra režģis**

3.4. Spēles vadības problēmas pētīšana

Lielākai daļai mūsdienu mobilajiem tālruņiem nav fiziskās tastatūras. Tās vietā tiek izmantots skārienjūtīgais ekrāns ar dažām fiziskām vai sensora pogām. Atšķirībā no video spēlēm, kur vienmēr ir kursorsvira vai tastatūra, mobilo spēļu programmētājiem jādomā par spēles vadības realizāciju pašiem, un Windows Phone 7 platforma nav izņēmums.

Windows Phone 7 viedtālruņiem ir 3 vadības pogas (3.4. attēls):

- **„Back”** – sistēma un programmas izmanto šo pogu navigācijas kontrolēšanai. Šī poga atgriež lietotāju uz iepriekšējo ekrānu.
- **„Start” –** atgriež lietotāju uz pamatvirsmas ekrānu.
- **„Search” –** meklēšanas iespējas sistēmā.

3.4. att. **Windows Phone 7 vadības pogas**

Spēles vadībai šīs pogas nav rekomendēts pielietot, izņemot atgriešanas pogu, kuru iespējams pielietot spēles izvēlnēs.

Pastāv vairākas iespējas realizēt spēles vadīšanu:

1. **Ekrāna kursorsvira**

Katrā spēlē, atkarībā no spēles vadības vajadzībām, kursorsvira varētu būt realizēta savādāk (3.5. attēls). Izstrādātājs, zinot nepieciešams spēles darbības, atstāj tikai vajadzīgās pogas. Parasti kursorsviras tiek realizētas spēlēs ar horizontālu ekrāna orientāciju – tādā gadījumā viedtālruni ir ērtāk turēt rokās– un tās izskatās pēc parastas kursorsviras.

3.5. att. **Ekrāna kursorsviras**

Nopietnākās ekrāna kursorsviras problēmas ir, pirmkārt, ka tās ir virtuālas un spēlētājs nejūt, vai viņš spiež uz pogu vai trāpa garām, un otrkārt, kursorsvira aizņem lielu ekrāna daļu un, spēlējot, lietotājs ar pirkstiem pats sev aizklāj šo ekrāna daļu. Šo problēmu iespējams daļēji novērst, realizējot pēc iespējas ērtākas vadības pogas – izvēloties un notestējot optimālākas kursorsviras pozīcijas un krāsas, kā arī ņemt to vērā spēles loģikas realizācijā. Pastāv speciālas ekrāna kursorsviras – mazas ierīces ar pogām vai bez tām, kuras iespējams piesaistīt virtuālām pogām. Tas atrisina tikai jūtamības problēmu.

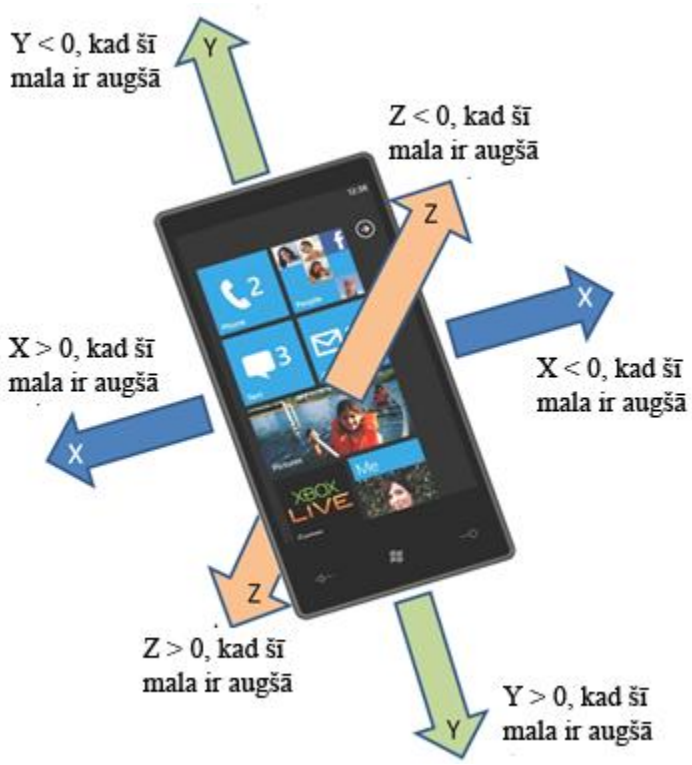
2. **Skārienjutīgais ekrāns**

Skārienjutīgais ekrāns atbalsta vairākus pirkstu pieskārienus vienlaicīgi. Windows Phone 7, kā arī citas mobilās platformas, ļauj programmētājiem atšķirt un pielietot žestus. Windows Phone 7 atbalsta dažāda tipa žestus – nospiešana, dubultnospiešana, turēšana,

dažāda tipa vilkšana, sašaurināšana/izstiepšana ar vairākiem pirkstiem, kā arī iespējams realizēt savus žestus [1, p. 40]. Pielietojot noteikta tipa žestus, iespējams paplašināt spēles vadīšanas iespējas.

3. Akselerometrs

Akselerometrs ir maza ierīce, kura mēra paātrinājumu. Kad tālrunis atrodas mierā stāvoklī, ar akselerometra palīdzību iespējams noteikt, kur atrodas ziemeļpols. Ar akselerometra palīdzību iespējams noteikt, kā tiek pagriezts telefons un cik ātri. Programmētājām ir iespēja nolasīt akselerometra stāvokli, kas tiek noteikts ar trīs dimensijas vektoru (X, Y, Z), kur katra vērtība ir no -1 līdz 1. Vektora stāvoklis atkarībā no viedtālruņa pozīcijas telpā ir aprakstīts 3.6. attēlā [2, p. 80].



3.6. att. **Akselerometra vektora stāvokļa apraksts**

Spēlēs akselerometru iespējams izmantot kā kursorsviru – vēršot viedtālruni uz vajadzīgo pusi. Vienīga atšķirība ir tas, ka akselerometram ir neliela aizture un dinamiskajās spēlēs tas ir jāņem vērā.

4. Citi varianti

Programmētājam ir iespēja pielietot citu viedtālruņa esošu aparatūru nestandarta vadīšanas realizācijai: mikrofonu, kameru vai GPS. Gaismas sensora stāvokli Windows Phone 7 operētājsistēmā programmētājiem nolasīt nav iespējams.

Kombinējot un pielietojot aprakstītus spēles vadības variantus, iespējams realizēt satriecošas spēles vadīšanas idejas. Kādu variantu izvēlēties ir atkarīgs no realizējamās spēles.

3.5. Skaņa un vibrācijas

Spēlē obligāti ir skaņas. Tās papildina spēli, padara to interesantāku un palīdz lietotājam orientēties spēlē. XNA ietvars ļauj ērti strādāt ar .wav, .mp3 un .wma audio formātiem. Skaņas iespējams atskaņot ciklā, mainīt vairākus atskaņošanas parametrus un pielietot 3D efektus.

Viedtālruņos ir vibrācijas ierīce, un XNA ietvars ļauj programmētājiem manipulēt arī ar šo ierīci - ieslēgt vibrāciju uz noteikto laiku un izslēgt. Vibrāciju iespējams pielietot arī kā papildelementu spēlē.

Nepareizi izvēlēta skaņa vai vibrācijas lietošana varbūt kairinoša, tāpēc obligāti jāpadomā, cik bieži un kad to lietot, un jārealizē to atslēgšanas opcijas.

3.6. Sadursmes

Katra spēle satur vairākus objektus, un viens no galvenajiem spēles tehniskajiem uzdevumiem ir realizēt sadursmes starp tiem. Pārklāšanas noteikšana tiek realizētā gandrīz jebkurā spēlē. Tā ir nepieciešama, lai spēle varētu noteikt brīdi, kad viens objekts saduras ar otru, vai kad viens objekts ir šķērsojis noteikto pozīciju un spēlei jāreaģē uz to. Gadījumos, kad spēlē ir daudz objektu, kuri mijiedarbojas viens ar otru, pārklāšanas noteikšanai jābūt realizētai teicami, savādāk lietotājs ieraudzīs spēles „bremzēšanu”.

Pirms domāšanas, kā noteikt kolīzijas, nepieciešams padomāt par to, kas notiks divu objektu sadursmes brīdī. Vieglākais risinājums būtu iznicināt vienu no objektiem vai arī pārvietot vienu vai abus objektus atpakaļ uz pozīciju pirms sadursmes. Otrais risinājums

nepieciešams gadījumos, kad tiek realizētas reālistiskas sadursmes (piemēram atlēcieni). Šajā gadījumā nepietiek noteikt tikai objekta sadursmes notikumu, bet arī nepieciešams noteikt, kurā punktā tā notika [23].

Eksistē daudz veidu, kā tiek realizēta sadursmes noteikšana spēlēs. Daži no tiem ir:

1. **Punktu pārbaude**

   Tiek salīdzināts katrs objekta punkts ar katru cita objekta punktu. Jā kādi no punktiem pārklājas, kolīzija ir atrasta. Šis ir pats lēnākais algoritms, un parasti tas netiek izmantots.

2. **Apvilktās figūras**

   Sameklējot katram objektam primitīvu ģeometrisku figūru – apli, trīsstūri vai taisnstūri (3D spēlēs figūras ir telpiskas), kurā iespējams ierakstīt objekta tekstūru (3.7. attēls), pārbaudīt, vai nenotiek kolīzija starp figūrām. Ja notiek, jāpārbauda katrs punkts pēc pirmā algoritma, vai arī uzreiz tiek noteikts, ka objekti pārklājas (gadījumā, ja objekta tekstūra ir primitīva vai ir līdzīga primitīvai figūrai un spēle ir dinamiska).

3.7. att. **Apvilktās objektu tekstūras**

Gadījumos, kad objekta tekstūrai ir sarežģīta forma, iespējams modificēt šo metodi ar dažādiem paņēmieniem:

- sadalīt tekstūru uz vairākām primitīvām figūrām un pielietot vairākas iterācijas pārbaudei,
- pielietot ierobežojoša tilpuma hierarhijas struktūru [18] – šī struktūra ir koka struktūra, kura satur sakārtotas ģeometriskās figūras. Visas objektu formas ir koka lapas, bet mezgli ir figūras, kuras ierobežo apakšā esošās figūras. Ja objekta figūra nepārklājas ar mezgla figūru, tā nevar pārklāties ar apakš esošajām figūrām.

- apvilkt tekstūru ar daudzstūri un realizēt pārklāšanas noteikšanu daudzstūriem. Eksistē vairāki algoritmi darbam ar daudzstūru kolīzijām [3], daži no tiem ir:
    - J. O'Rourke algoritms (tikai izliekti daudzstūri);
    - Leonova algoritms;
    - Līnij mezglu algoritms;
    - Triangulācijas algoritmi.

  Šādi algoritmi prasa vairāk resursu un, ja tās ir iespējams, ātrākai apstrādei ir vērts izmantot tikai izliektus daudzstūrus, pielietojot teorēmu par sadalošu asi [17].

Apvilktās figūras risinājums ir populārākā metode, kura ļauj ātri atmest nevajadzīgos objektus, neiedziļinoties objekta detaļās. Šīs metodes realizācija var stipri atšķirties atkarībā no tekstūra kontūra sarežģītības, sadursmes tipiem, objekta izmēriem un ātrumiem kā arī nepieciešamā precizitātes līmeņa.

3. Ekrāna sadalījums

Pirms pārklāšanas pārbaudes spēles ekrāns tiek sadalīts zonās. Zonas izmērs tiek izvēlēts atkarībā no lielākā objekta izmēra. Pirms kolīzijas pārbaudes objektam tiek noteikta zona, kur tas pašlaik atrodas, un pārbaudei ir nepieciešams pārlūkot tikai blakusesošas zonas, izmantojot, piemēram, iepriekšējo metodi. 3D spēlēs iespējams izmantot bināro telpas sadalījumu [19] – atspoguļot zonas koka struktūrā, sakārtojot zonas pēc attāluma no dotā objekta.

Metodi ir vērts pielietot, ja objektu skaits uz ekrāna ir liels un spēles pasaules izmērs ir liels.

Sadursmes noteikšanā jāņem vēra, ka var notikt situācijas, kad mazs dinamisks objekts „izies cauri" citam objektam, kas ir kļūda spēlēs realizācijā. Šāda situācija varētu notikt aiztures dēļ – pirms kolīzijas pārbaudes mazs objekts tuvojās citam objektam, notiek negaidīta aizture un nākamais spēles pārbaudes cikls notiek pēc tām, kad mazs objekts jau ir pārlidojis to objektu un

kolīzija nepamana šo pārlidojumu. Šīs problēmas novēršanai arī var pielietot vairākus risinājumus:

- pārbaudīt nevis divus objektus, bet izveidot no tiem lielākus objektus, ņemot vērā objektu izmērus, kustības trajektorijas un ātrumus,
- dinamiskiem objektiem pārbaudīt vairākus punktus – no punkta pašreizējas pozīcijas līdz iepriekšējai pozīcijai izvēloties vairākus punktus.

4. SPĒLES REALIZĀCIJA UN PROBLĒMAS RISINĀJUMS

Apskatot Windows Phone 7 platformu, XNA spēles ietvara iespējas un spēles realizācijas problēmas nākamais solis ir izveidot spēli, lai iegūtu praktiskās zināšanas par spēles veidošanu jaunajā platformā. Spēles realizācijai tika izdomāta spēle, kurā būs realizētas pamatelementi – dinamiskā spēles vadība, sadursmes realizācija, skaņas un izvēlnes. Spēlei jābūt piemērotai mobilajām ierīcēm, ar saprotamu saskarni, ērtu vadāmību un gatavai izplatīšanai.

Spēles realizācijas gaitā tika atrisinātas apskatītās problēmas: animācijas realizācija, piemērotās spēles vadības realizācija kā arī optimālais sadursmes noteikšanas algoritms.

Spēles izstrādei, balstoties uz secinājumiem, kuri tika iegūti 3.1. sadaļā, bija izvēlēta XNA tehnoloģija. Autors izmantoja Visual Studio 2010 Ultimate izstrādes vidi ar Windows Phone Developer Tools 7.0 rīkkopu, kura ietver nepieciešamus rīkus spēles realizācijai: Windows Phone emulatoru un XNA Game Studio 4.0 rīkkopu.

Spēles realizācija tika sadalīta vairākos etapos:

1. Spēles sižets un ideja;
2. Grafiskie objekti un animācijas problēmas risinājums;
3. Spēles objektu un pamatfunkciju izstrāde;
4. Spēles vadība;
5. Sadursmes problēmas risināšana;
6. Spēles beidzamas funkcijas realizācija: informatīva josla, skāņas un izvēlnes;
7. Testēšana;
8. Spēles izplatīšana.

Pēc spēles realizācijas tika apskatītas platformas izstrādes nianses un tika definēti nākotnes plāni.

4.1. Spēles sižets un ideja

Spēles nosaukums ir „DiveUp”. Spēle ir 2D viena spēlētāja arkādes tipa spēle, kurā burbulis, pārvietojoties uz augšu, satiekas ar dažāda tipa šķēršļiem. Spēlētājs, kontrolējot burbuli, pārvietojas pa ekrānu un izvairās no šķēršļiem. Spēlētājs varēs izšaut pa šķēršļiem, lai tos iznicinātu, kā arī eksistē neiznicināmie šķēršļi. Šaušanai jābūt diviem režīmiem – parastais šāviens un speciālais šāviens, kurš iznicinās visu sava lidojuma trajektorijā. Spēlētājam būs sava

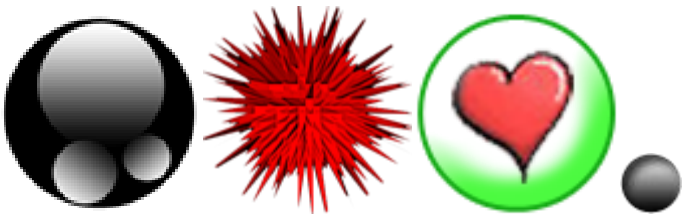
dzīves skala, kura samazinās, satiekoties ar šķēršļiem. Dzīves daudzumu būs iespējams papildināt.

Spēles ideja – sakrāt pēc iespējas vairāk punktus, kuri tiek pieskaitīti par spēlēšanas laiku un iznicinātiem šķēršļiem.

4.2. Grafiskie objekti un animācijas problēmas risinājums

Nākamais etaps spēles izstrādē ir realizēt grafiskus objektus. Spēlē būs pieejams tikai vertikālais ekrāna režīms, tāpēc ka burbulis peldēs uz augšu un spēlētājam būs nepieciešams redzēt vairāk objektus, kuri tuvojas no augšas.

Spēlē ir 4 tipu objekti: pats burbulis, ienaidnieka modelis, dzīves papildināšanas objekts un lodes. Objektu grafiskai realizācijai tika izveidoti PNG formāta attēli (4.1. attēls) ar caurspīdīgo fonu. Attēla izmērs tika izvēlēts lielāks nekā nepieciešams, lai nākotnē būtu iespējams detalizēt vai arī palielināt objektus.



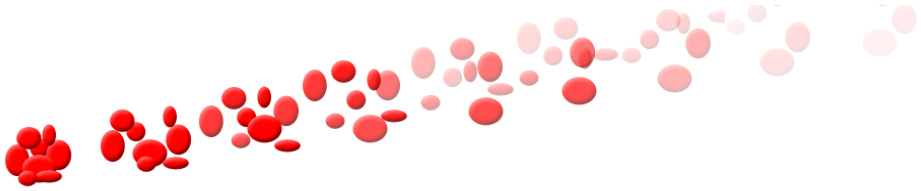
4.1. att. **Spēles grafiskie objekti**

Animācijas problēmas risināšanai nepieciešams atrast sabalansētu risinājumu – uzlabot spēles izskatu, nepārslogot ar dinamiskie objektiem un nepārslogot viedtālruņa aparatūru. Šīm nolūkam tika analizēts cik bieži un kādas darbības un objektus redzēs lietotājs.

Burbulis un ienaidnieki ir visvairāk sastopami objekti spēlē un objektu animācijai ir izmantotas transformācijas iespējas – ar lielu ātrumu nedaudz mainot objekta izmēru. Šāda veidā animācija neprasa daudz resursus un ir ļoti viegli realizējama. XNA ietvarā eksistē standarta metode SpriteBatch.Draw, kura atļauj dažādos veidos izvadīt tekstūru uz ekrāna – noteiktā pozīcijā, krasā un mērogā, noteikto tekstūras daļu.

Biežāk sastopama darbība būs ienaidnieka iznicināšana, tāpēc šai darbībai ir realizēta animācija, pielietojot gariņu tehnoloģiju – horizontāli salīmējot kadru pēc kadra, tiek realizēta

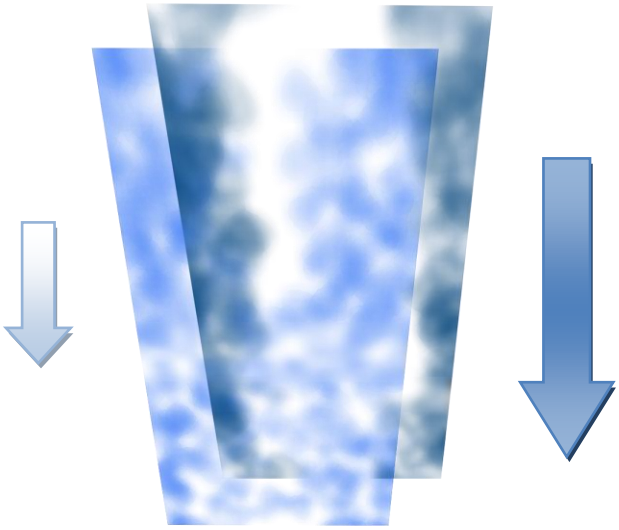
objekta izjaukšana (4.2. attēls). Animācijai nav jābūt ļoti detalizētai, jo tā būs redzama tikai īsu brīdi.



4.2. att. **Gariņš objekta izjaukšanai**

Animācijas realizācija balstās uz piemīnētas metodes Draw, pielietojot kuru iespējams atspoguļot tekstūras daļu [31]. Atšķirība no statiskiem objektiem, animācijas objektiem ir nepieciešams zināt cik laika ir pagājis no iepriekšējas zīmēšanas, lai izvadīt pareizo kadru, tāpēc visas animācijas objektus, nepieciešams atjaunot katrā spēles cikla atjaunošanās fāzē.

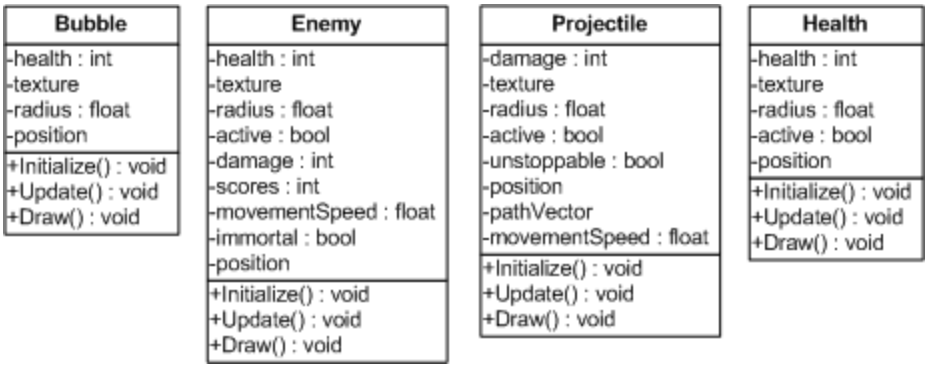
Nākamais solis ir realizēt ekrāna pārvietošanas efektu. Šāda efekta realizācijai nav nepieciešams realizēt sarežģītu objektu pārvietošanu, to nodrošinās paralakses ritināšanas (*parallax scrolling*) efekts [20]. Šīs efekts izmanto vairākas fona bildes, kuras pārvietojas ar dažādu ātrumu – zemāk esošā fona bilde pārvietojas lēnāk nekā virsu esošais fons. Redzot šādas pārvietojamas bildes, rodas ilūzija, ka bilde ir apjomīga un pārvietojas (4.3. attēls).



4.3. att. **Paralakses ritināšanas efekts**

4.3. Spēles objektu un pamatfunkciju izstrāde

Pamatobjektu realizācijas etapā tika izstrādāta pamatobjektu UML klases diagramma (4.4. attēls). Realizējot objektus C# programmēšanas valodā un pielietojot emulatoru un viedtālruni tika saskaņoti objektu izmēri un sākumstāvokli.



*4.4.att.***Pamatobjektu klašu diagramma**

4.4. Spēles vadības realizācija

Spēle atpazīst vairākas darbības – burbuļa pārvietošana uz dažādām pusēm un šaušana divos režīmos.

Tika apskatīti vairāki varianti:

- Kursorsviras realizācija ar šaušanas pogu

  Tā kā spēle būs pieejama tikai vertikālā režīmā, kursorsviru nav iespējams ērti pielietot un šīs variants tika izslēgts.

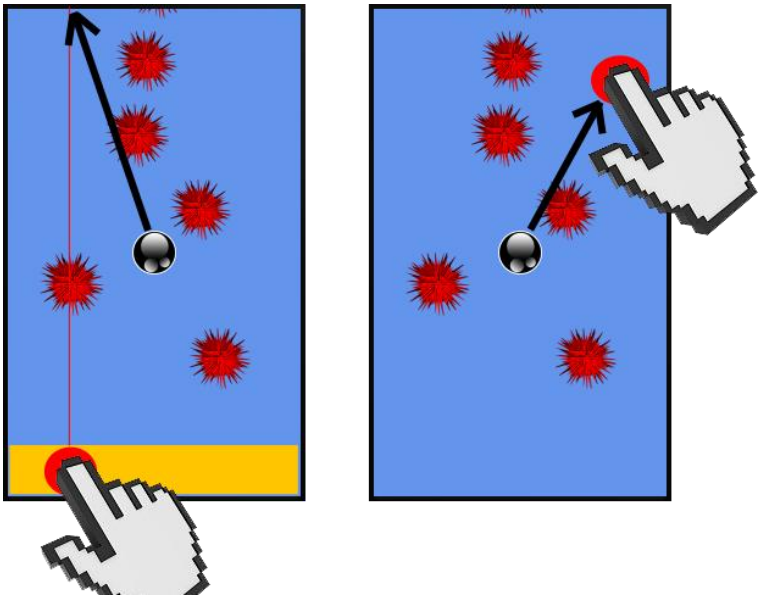
- Akselerometra un skārienjūtīga ekrāna kombinācijas pielietošana

  Burbuļa kontrolēšanai ideāli der akselerometra izmantošana. Kaut arī pierast pie akselerometra izmantošanas aizņem kādu laiku, tomēr lietotājs pēc pielāgošanas ļoti veiksmīgi to izmanto.

  Šaušanas realizācija ir nopietnākā problēma, jo ir nepieciešams atšķirt divus šaušanas veidus. Viens īsais ekrāna pieskāriens ideāli der parastajam šāvienam. Vieglākai izmantošanai iespējams atstāt ekrānā apakšā nelielu laukumu, nospiežot pa kuru notiek šāviens (4.5. attēls, kreisajā pusē). Cits variants ir, ka šaušana notiek uz

punktu, uz kuru spēlētājs ir nospiedis (4.5. attēls, labajā pusē). Pielietojot pirmo variantu, lietotājs vienmēr redzes ekrānu, bet šaušanas precizitāte būs mazāka un nebūs iespējams izšaut jebkurā pusē. Otrajā gadījumā lietotājs varēs izšaut ļoti precīzi, bet ar savu roku viņš aizsegs ekrānu.

Šaušanas precizitāte ir kritiskāka, un tāpēc tika realizēts otrais variants – lode lido uz pieskāriena vietu. Speciālā šaušana tika realizēta, pielietojot standarta turēšanas žestu „Hold" - turot pirkstu vienā punktā ilgāk, tiek veikts speciāls šāviens.

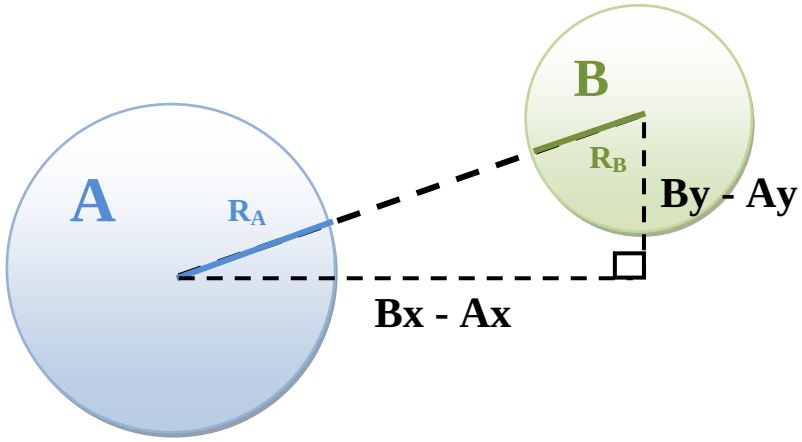


*4.5.att.***Šaušanas vektora varianti**

4.5. Sadursmes problēmas risināšana

Pirms sadursmes realizācijas tika analizēti sadursmes tipi un objektu figūras. Spēlē visiem objektiem ir apaļa forma un nav jāzina, no kuras puses notika sadursme. Tāpēc tika konstatēts, ka sadursmes realizācijā būs pielietota apvilktās figūras metode, kura ir apskatīta 3.6. sadaļā.

Riņķa sadursmes algoritms ir sekojošs: zinot divu objekta centra pozīcijas un pielietojot Pitagora teorēmu, tiek aprēķināta distance starp objektiem (4.6. attēls). Gadījumā, ja šī distance ir mazāka par objektu rādiusu summu, objekti pārklājas.



4.6.att. **Attālums starp riņķiem**

Apskatīsim šo mazu algoritmu sīkāk. Pieņemsim, mums ir divas riņķa līnijas A un B. A riņķa līnijas centra koordināte (Ax, Ay) un rādiuss R_A. B riņķa līnijas centra koordināte (Bx, By) un rādiuss R_B.

1. Algoritms sadursmes noteikšanai:

   a. Distance starp riņķiem = $\sqrt{(Bx - Ax)^2 + (By - Ay)^2}$;
   b. Ja distance starp riņķiem ir lielāka par $R_A + R_B$, tad riņķi nesaskārās, savādāk ir notikusi sadursme.

Pat tādu mazu algoritmu iespējams optimizēt: kā zināms kvadrātsakne ir darbietilpīga operācija. Otrais algoritma variants neizmanto to:

2. Algoritms bez kvadrātsaknes:

   a. Distance starp riņķiem kvadrātā = $(Bx - Ax)^2 + (By - Ay)^2$;
   b. Ja distance starp riņķiem kvadrātā ir lielāka par $(R_A + R_B)^2$, tad riņķi nesaskārās, savādāk ir notikusi sadursme.

Atjaunotā algoritmā nav darbietilpīgas kvadrātsaknes operācijas, tā vietā ir viena reizināšana. Arī šo algoritmu iespējams optimizēt: nerēķinot distanci uzreiz, bet pārbaudīt vai apvilktie kvadrāti saskārās.

3. Algoritms ar pirmspārbaudi:

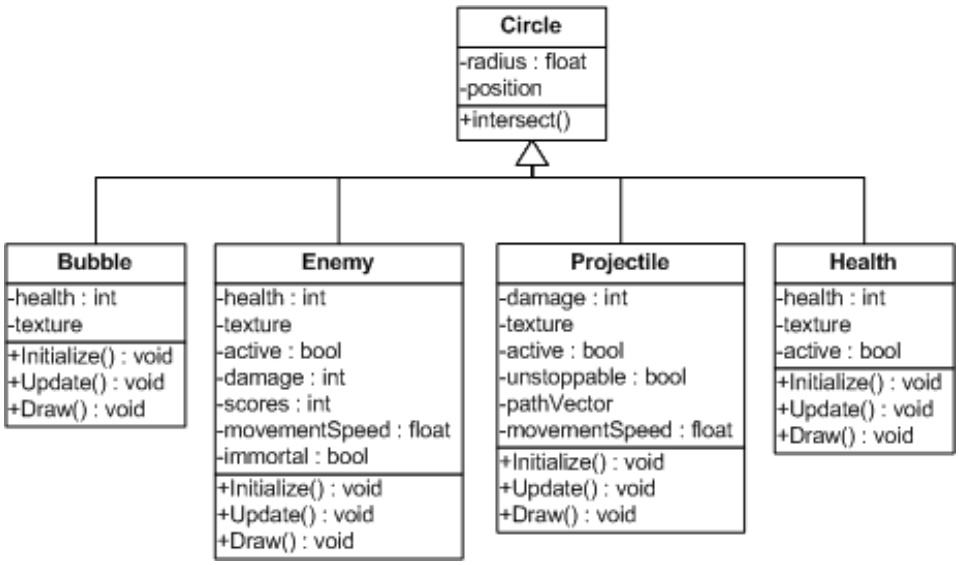
    a. Aprēķināt A riņķa līnijas kvadrāta izmērus (zinot centru un rādiusu, kvadrāta koordinātes būs: $X1 = Ax - R_A$, $X2 = Ax + R_A$, $Y1 = Ay - R_A$ un $Y2 = Ay + R_A$);
    b. Aprēķināt B riņķa līnijas kvadrāta izmērus;
    c. Ja kvadrāti pārklājas pārbaudīt riņķa līnijas pārklāšanas ar otro algoritmu.

Šīs algoritms ļauj ātrāk pārbaudīt objektus, kuri ir tālu viens no otra, šādiem objektiem netiek izmantota pat reizināšanas operācija, tikai saskaitīšana, atņemšana un loģiskās operācijas. Savukārt objektiem kuri pārklājas rēķināšana nedaudz palielinās.

Spēlē objektu skaits kurš nesaskārās ir vairāk nekā pārklāšanas objektu (gadījumos, kad pārklāšana ir notikusi, objekts tiek iznicināts), tāpēc spēlē tika pielietots pēdējais algoritms.

Tā kā objekti nav ļoti dinamiski un ir mazi – gadījums, kad viens objekts izies cauri otram, nav iespējams. Par katru sadursmi spēlētājs saņems definēto punktu skaitu.

Realizētās pamatobjektu klasēs tika izdarītas izmaiņas: uzrakstīta jaunā riņķa līnijas klase (2. pielikums) un katrs apaļš objekts manto šo klasi (4.7. attēls). Šāda veidā katram objektam ar apaļu formu būs pieejama sadursmes pārbaudes metode, kura saņemot citu apaļu objektu, aprēķinā vai ir notikusi kolīzija ar to vai nav. Veicot šādas izmaiņas realizācijā, nākotnē būs viegli pievienot cita tipa objektus un kolīzijas noteikšanai būs jāuzraksta vienu metodi.



4.7.att. **Pamatobjektu atjaunotā klašu diagramma**

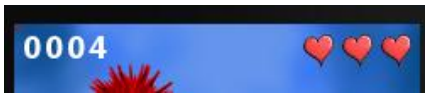
4.6. Spēles beidzamas funkcijas realizācija

- **Skaņas un vibrācijas**

Skaņas atskaņošanai un vibrācijas pielietošanai XNA ietvarā ir standarta funkcijas. Galvenā nianse, tika realizēta šo parametru izslēgšanas/ieslēgšanas iespēja opciju sadaļā.

- **Spēles informatīva josla**

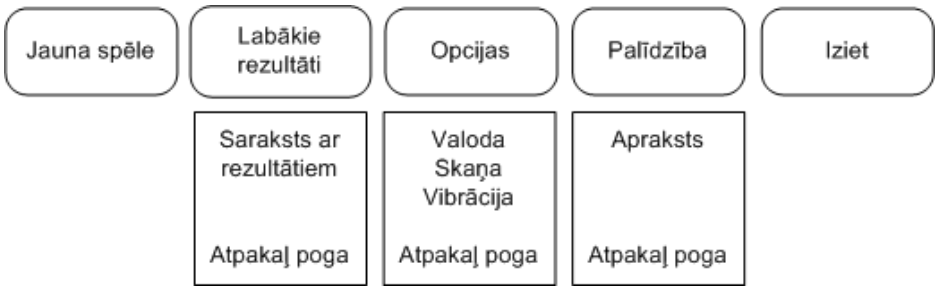
Spēlētājām nepieciešams redzēt pašreizējo spēles stāvokli – sakrāto punktu daudzumu un dzīves līmeni. Analizējot esošo spēles stāvokli, ekrāna augšējā pusē tika izveidota informatīvā daļa ar ciparveida punktiem un dzīves līmeni (4.8. attēls). Līmenim tika pielietota grafiska reprezentācija, jo burbuļa dzīves līmenis nevar būt liels skaitlis. Savukārt punktiem ir pielietota XNA ietvara standarta metodi DrawString – teksta izvadīšanai. Šai metodei, tāpat kā jau apskatītai metodei Draw, ir vairāki ieejas parametri – ielādēta fonta objekts, pozīcija, krāsa un citi.



4.8.att. **Spēles informatīva josla**

- **Spēles izvēlne**

XNA tehnoloģija nav pieejamas operētājsistēmas standarta elementi, tāpēc visas izvēlnes nepieciešams realizēt programmētājiem. Dokumentācijā ir pieejams piemērs izvēlnes realizācijai XNA spēlē [26]. Pielāgojot šo izvēlni savam vajadzībām, tika realizēta spēles izvēlne ar divām valodām: angļu un latviešu (4.9. attēls).



4.9.att. **Spēles izvēlnes struktūra**

Visus iestatījumus ir nepieciešams saglabāt, lai atvērot spēli nākamo reizi, tie būtu atjaunoti. Šīm nolūkam tika izmantota pieejama izolēta noliktava. Izolēta noliktava garantē, ka pieeja pie tā ir tikai vienīgai lietojumprogramma. Strādāt ar šo noliktavu iespējams, ka ar parasto failu [27].

Izejot no spēli vai zaudējot, lietotājam ir piedāvāts saglabāt rezultātu, ievadot savu vārdu. Teksta ievadīšanai XNA ietvarā ir pieejama Windows Phone 7 standarta ekrāna tastatūra (metode Guide.BeginShowKeyboardInput). Labākie rezultāti būs pieejami rezultātu sarakstā. Labāko rezultātu glabāšanai arī ir pielietota izolēta noliktava.

4.7. Testēšana

Spēles kvalitātes pārbaudei tika definēts testēšanas plāns:

1. Pielietojot koda analīzes rīku, atrast problemātiskas vietas pirmkodā un izlabot tos;
2. Notestēt spēles metodes pielietojot vienībtestus;
3. Uzrakstot spēles pielietošanas scenārijus notestēt spēli uz emulatora un viedtālruņa;
4. Veikt spēles testēšanu reālajos apstākļos ar ierīci un dažādiem spēlētājiem.

4.7.1. Koda analīze

Koda analīze tiek veikta neizpildot lietojumprogrammu, analīzes gaitā tiek pārbaudīts lietojumprogrammās pirmkods. Visual Studio izstrādes vidē ir pieejami koda analīzes rīki. Koda analīzei iespējams izvēlēties analīzes līmeni – sākot no minimālām prasībām, beidzot ar pilno pārbaudi.

Pielietojot realizētai spēlei koda analīzes rīku ar maksimālo pārbaudes līmeni, tika atrasti un izlaboti vairāk par 100 brīdinājumiem, vairāki no tiem bija brīdinājumi par neizmantotiem mainīgiem un nepareizo simbolu virknes salīdzināšanas pielietošanu. Tās sekmēja pirmkoda uzlabošanai.

4.7.2. Vienībtestēšana

Vienībtestēšana ir testēšana metode pirmkoda pārbaudei. Visual Studio vidē ir iebūvētās vienībtestēšanas iespējas. Izvēloties kādas klases un metodes nepieciešams pārklāt ar testiem, tika saģenerēts vienībtestēšanas karkass, kuru nepieciešams papildināt ar testiem. Spēlē kritiskākā metode ir sadursmes notikuma pārbaude, kurai pirmkārt tika realizēta testēšana (4.1. tabula). Metode atgriež 1 ja objekti pārklājās, savādāk 0.

4.1. tabula

Sadursmes metodes testēšanas tabula

Grafiskais apraksts	Apraksts	Sagaidāmais rezultāts	Testēšanas rezultāts
	Vienāda izmēra objekti nepārklājās	0	0
	Dažāda izmēra objekti atrodas ļoti tuvu viens otrām un nepārklājās	0	0
	Objekti pārklājas minimāli	1	1
	Objekts ir citā objektā	1	1
	Divi vienādi objekti atrodas viens virs otra	1	1
	Viens no objektiem daļēji atrodas aiz ekrāna, objekti nepārklājās	0	0
	Objekti pārklājas aiz ekrāna robežām	1	1
	Viens no objektiem daļēji atrodas aiz ekrāna, objekti pārklājās	1	1

Vienībtestēšanas piemērs ir pieejams pielikumā 1. Balstoties uz iegūtiem rezultātiem vienībtestēšana bija sekmīgi paveikta.

4.7.3. Spēles testēšana

Pēc katra izstrādes etapa spēle tika pārbaudīta uz emulatora un reālas ierīces. Beidzamā spēles testēšana, nelielā spēles izmēra dēļ, netika veidoti testēšanas scenāriji, bet testēšana notika vienkārši spēlējot uz reālās ierīces. Spēli testēja arī spēlētāji, kuri nepiedalījās spēles izstrādē.

Testēšanas laikā uz reālas iekārtas tika atklāti daži trūkumi:

- Spēlējot spēli un nešaujot pa ienaidniekiem, viedtālrunis automātiski izslēdzas pēc noteikta laika – parejot uz gaidīšanas režīmu. Šīs mehānisms ir standarta mehānisms, kurš saglabā viedtālruņa baterijas enerģiju, izslēdz ekrānu un ieslēdz gaidīšanas režīmu.

  Problēmas risināšanai tika atslēgt automātisku gaidīšanas režīms nomainot vērtību UserIdleDetectionMode mainīgajām [28].
- Spēles procesa problēma. Spēlētājs ātri atrod vietu uz ekrāna, kurā ir ļoti viegli krāt punktus un nav nepieciešams daudz pārvietoties, pielietojot akselerometru. Šīs problēmas risināšanai tika pielabots spēles process: ciklā, pēc noteiktā laika, spēlētājam vizuāli tika paziņots par nepieciešamību nobīdīties no ekrāna malām un pēc dažām sekundēm malās paradīsies ērkšķi, kuri noņems burbulim enerģiju, ja spēlētājs nepārvietosies tuvāk centram.

Pabeidzot spēles izstrādi un testēšanu, spēles bija sekmīgi pabeigta (3. pielikums spēles cikla pirmkoda daļa) (4.10. attēls). Saņemta atsauksme par izstrādāto spēli testēšanas laikā ir pozitīva. Spēles pirmkods satur vairāk par 2000 koda rindiņām.

4.10.att. **Spēles ekrānuzņēmumi**

4.8. Spēles izplatīšana

Windows Phone 7 platformai ir vienotais veikals, kurā tiek apkopotas visas lietojumprogrammas un spēles. Par dalību tajā nepieciešams samaksāt 99$, bet DreamSpark programmas studentiem piedalīšanas tajā ir bezmaksas (ir ierobežojumi lietojumprogrammu skaitam). Saņemtā peļņa no programmatūras izplatīšanas tiek sadalīta: 30% patur *Microsoft* un 70% saņem izstrādātājs

Programmatūras izplatīšanai nepieciešams izdarīt sekojošas darbības:

1. Jāreģistrējas veikalā, ievadot personas datus;
2. Pēc reģistrācijas notiek identitātes pārbaude – izstrādātājam jānosūta savus noskenētus dokumentus *Microsoft* partnerim *GeoTrust*.
3. *GeoTrust* pārbauda dokumentus un veiksmes gadījumā, aktivizē reģistrāciju.
4. Aktivizētiem lietotājiem iespējams augšupielādēt savu programmatūru.
5. Augšupielādējot lietojumprogrammas vai spēles, notiek pārbaude.
6. Pēc veiksmīgas pārbaudes programmatūra ir pieejama visiem Windows Phone 7 lietotājiem.

Reģistrācijas laikā, autors ir atklājis problēmu - veikalā ir ierobežots dalībvalstu skaits un piereģistrēties no Latvijas nav iespējams. Pēc problēmas analīzes, tika izdarīti neapmierinoši secinājumi:

- Piedalīties oficiālajā veikalā izstrādātājam no Latvijas nav iespējams, un kad būs šāda iespēja – nav zināms.
- Eksistē starpnieks *Yalla Apps*, ar kura palīdzību iespējams augšupielādēt savas lietojumprogrammas. Starpnieks patur 20% no izstrādātāja peļņas. Šis serviss arī nav pieejams bezmaksas reģistrācijai studentiem no Latvijas.

Spēles izplatīšanas problēmas palika neatrisināta veikala reģionāla ierobežojuma dēļ.

4.9. Platformas un spēles izstrādes nianses

Strādājot ar jaunāko Windows Phone 7 operētājsistēmu un rakstot darbu, autors konstatējis vairākas izstrādes nianses, kurus ir vērts ņemt vērā arī citiem izstrādātājiem:

- Ekrānuzņēmumus no viedtālruņa izdarīt nav iespējams. Iespējams uztaisīt ekrānuzņēmumu no emulatora, sakonfigurējot nepieciešamu izmēru, vai nofotografēt viedtālruņi;
- Programmētājiem nav pilnas pieejas pie fotokameras – ir iespēja nofotografēt un strādāt ar šo fotogrāfiju, bet reāla laika video dabūt nav iespējams (tiks izlabots nākamajos Windows Phone atjauninājumos);
- Problēmas ar standarta fontiem ar burtiem, kas nav angļu alfabētā (jāmeklē piemērotāks fonts);
- Izstrādes vides pilnvērtīgai darbībai ir nepieciešama Windows Vista vai Windows 7 operētājsistēma ar DirectX 10 saderīgu videokarti;
- Izmantojot vibrāciju, vienmēr jāatstāj lietotājām iespēju to atslēgt;
- Ir uzmanīgi jāveido jauni objekti spēlēs ciklā, ierīces atmiņa ir mazāka nekā datoriem. Windows Phone 7 lietojumprogrammas un spēles izmanto tīrīšanas mehānismu (*Garbage Collection)*, kurš pēc noteiktā laika iznicina neizmantotus objektus. Iznicināšana spēlēs ciklā, varētu aizņemt daudz resursu un cikls aizkavēsies un spēlē notiks lēciens vai aizture [32]. Izplatītākas kļūdas piemērs ciklā simbolu virknēm pielietot konkatenācijas

operāciju, kura nepamanāmi veido jaunu objektu. Šīm nolūkam eksistē klase StringBuilder ar kuras palīdzību var rediģēt un papildināt simbolu virknes kļūdaini neveidojot jaunus objektus.

4.10. Nākotnes plāni

Pabeidzot spēles realizāciju, tika definēti sekojošie globālie nākotnes plāni:

- Sagaidīt vai atrast citu iespēju spēles publikācijai Windows Phone 7 programmatūras veikalā.
- Sekot līdzi jaunajām iespējām nākamajos Windows Phone 7 platformas atjauninājumos.
- Pārnest spēli uz iOS platformu un paplašināt spēli.
- Turpināt pilnveidoties mobilo spēļu un lietojumprogrammu izstrādes jomā.

Izstrādātas spēles uzlabošanas plāns:

- paplašināt spēli ar dažādiem ienaidnieku tipiem,
- papildināt spēli ar dažāda tipa kartēm;

Atkarība no gūtiem rezultātiem spēles izplatīšanā tiks nolemts par tālāko spēles pilnveidošanu.

5. REZULTĀTI UN SECINĀJUMI

Darba rezultātā tika izpētīta jaunākā *Microsoft* kompānijas mobilā operētājsistēma Windows Phone 7, tās iespējas no lietotāja un izstrādātāja viedokļiem, iegūta pieredze mobilās spēles izstrādē, pielietojot XNA tehnoloģiju un C# programmēšanas valodu. Tika izpētītas un atrisinātas izplatītākās mobilo spēļu realizācijas problēmas.

Tas bija pirmais darba autora projekts spēļu izstrādes sfērā un programmēšanā, pielietojot Visual Studio izstrādes vidi, tāpēc tā realizācija aizņēma trīs mēnešus. Ļoti daudz laika tika veltīts XNA ietvara pieejamo iespēju un mobilās spēles izstrādes principu un problēmu pētīšanai. Rezultātā tika izstrādāta Windows Phone 7 spēle, kura ir gatava izplatīšanai. Tika atrisinātas visas darba laikā radušās problēmas, izņemot spēles izplatīšanas problēmu, kuru pagaidām nav iespējams atrisināt veikala ierobežojumu dēļ.

Izstrāde pētāmai platformai ir ērta, kaut arī emulatoram, ar kuras palīdzību notiek izstrādātas programmatūras testēšana, ir nelieli trūkumi. Pieejama rīkkopa palīdz izstrādātājam ātri paveikt darbu un satur vairākas papildiespējas: pirmkoda analīzes un vienībtestēšanas rīkus. Izmantotais XNA programmatūras ietvars satur nepieciešamās metodes, atvieglo un paātrina mobilās spēles izstrādi. Tehnoloģijas apmācībai ir pieejami vairāki informācijas avoti – gan Interneta resursi, gan grāmatas.

Jaunākā Windows Phone 7 mobilā operētājsistēma ir viena no perspektīvākajām un ātri attīstāmajām sistēmām un autors rekomendē iepazīties ar šo platformu katram programmētājam, kurš nodarbojas ar mobilo lietojumprogrammu un spēļu izstrādi.

IZMANTOTĀ LITERATŪRA

1. **Williams, C., Clingerman, G.** *Professional Windows Phone 7 Game Development. Creating Games Using XNA Game Studio 4.* USA: Wiley Publishing, 2011.
2. **Petzold, C.** *Programming Windows Phone 7*, Washington: Microsoft Press, 2010.
3. **Ченцов, О.В., Скворцов, А.В.** *Обзор алгоритмов построения оверлеев многоугольников*, факультет информатики Томского государственного университета, 2003.
4. *Viedtālrunis*. [tiešsaistes] – [atsauce 10.02.2011.]. Pieejams: http://lv.wikipedia.org/wiki/Viedt%C4%81lrunis.
5. *Top 8 Mobile OSs from 2008 to 2011.* [tiešsaistes] – [atsauce 13.02.2011.]. Pieejams: http://gs.statcounter.com/#mobile_os-ww-yearly-2008-2011.
6. *Symbian*. [tiešsaistes] – [atsauce 20.02.2011.]. Pieejams: http://en.wikipedia.org/wiki/Symbian.
7. **Ricker, T.** *RIP: Symbian.* [tiešsaistes] – [atsauce 20.02.2011.]. Pieejams: http://www.engadget.com/2011/02/11/rip-symbian/.
8. *iOS (Apple).* [tiešsaistes] – [atsauce 20.02.2011.]. Pieejams: http://en.wikipedia.org/wiki/IOS_(Apple).
9. *Mobile operating system.* [tiešsaistes] – [atsauce 20.02.2011.]. Pieejams: http://en.wikipedia.org/wiki/Mobile_operating_system.
10. *Gartner report: Mobile gaumes revenues worldwide.* [tiešsaistes] – [atsauce 04.03.2011.]. Pieejams: http://www.mobilewebgo.com/gartner-report-mobile-games-have-5-billion-revenues-worldwide.
11. **Hollington, J. D.** Report*: Games are 58% of App Store, puzzles top genre.* [tiešsaistes] – [atsauce 13.03.2011.]. Pieejams: http://www.ilounge.com/index.php/news/comments/report-games-are-58-of-app-store-puzzles-top-genre/.
12. *XNA Game Studio or Silverlight: Which Product is Right for Me?* [tiešsaistes] – [atsauce 20.03.2011.]. Pieejams: http://create.msdn.com/en-US/education/catalog/article/which_product_for_windows_phone.

13. *The Silverlight and XNA Frameworks for Windows Phone.* [tiešsaistes] – [atsauce 22.03.2011.]. Pieejams: http://msdn.microsoft.com/en-us/library/ff402528(v=vs.92).aspx.
14. *XNA Game Development (First Steps).* [tiešsaistes] – [atsauce 25.03.2011.]. Pieejams: http://www.progware.org/Blog/post/XNA-Game-Development-(First-Steps).aspx.
15. **Asokan, S.** *What is 3D Animation?* [tiešsaistes] – [atsauce 25.03.2011.]. Pieejams: http://www.buzzle.com/articles/what-is-3d-animation-how-is-it-different-from-2d-animation.html.
16. *Metro Design Language of Windows Phone 7.* [tiešsaistes] – [atsauce 15.03.2011.]. Pieejams: http://www.microsoft.com/design/toolbox/tutorials/windows-phone-7/metro/.
17. *Separating axis theorem.* [tiešsaistes] – [atsauce 08.04.2011.]. Pieejams: http://en.wikipedia.org/wiki/Separating_axis_theorem.
18. *Bounding volume hierarchy.* [tiešsaistes] – [atsauce 08.04.2011.]. Pieejams: http://en.wikipedia.org/wiki/Bounding_volume_hierarchy.
19. *Binary space partitioning.* [tiešsaistes] – [atsauce 08.04.2011.]. Pieejams: http://en.wikipedia.org/wiki/Binary_space_partitioning
20. *Parallax scrolling.* [tiešsaistes] – [atsauce 13.04.2011.]. Pieejams: http://en.wikipedia.org/wiki/Parallax_scrolling.
21. *Mono (software).* [tiešsaistes] – [atsauce 15.04.2011.]. Pieejams: http://en.wikipedia.org/wiki/Mono_(software).
22. *MonoGame*. [tiešsaistes] – [atsauce 15.04.2011.]. Pieejams: http://monogame.codeplex.com/.
23. *MonoXNA.* [tiešsaistes] – [atsauce 15.04.2011.]. Pieejams: http://code.google.com/p/monoxna/.
24. *Collision Detection and Response.* [tiešsaistes] – [atsauce 18.04.2011.]. Pieejams: http://www.metanetsoftware.com/technique/tutorialA.html.
25. **Gouveia, D.** *2D Camera with Parallax Scrolling in XNA.* [tiešsaistes] – [atsauce 25.04.2011.]. Pieejams: http://www.david-gouveia.com/2d-camera-with-parallax-scrolling-in-xna/ .

26. *Game State Management. Code sample.* [tiešsaistes] – [atsauce 25.04.2011.]. Pieejams http://create.msdn.com/en-US/education/catalog/sample/game_state_management.
27. *Introduction to Isolated Storage.* [tiešsaistes] – [atsauce 25.04.2011.]. Pieejams: http://msdn.microsoft.com/en-us/library/3ak841sy(v=vs.80).aspx.
28. *Idle Detection for Windows Phone.* [tiešsaistes] – [atsauce 25.04.2011.]. Pieejams: http://msdn.microsoft.com/en-us/library/ff941090(v=vs.92).aspx.
29. **Topolsky, J.** *Windows Phone 7 Series is official and Microsoft is playing to win.* [tiešsaistes] – [atsauce 28.04.2011.]. Pieejams: http://www.engadget.com/2010/02/15/windows-phone-7-series-is-official-and-microsoft-is-playing-to/?s=t5.
30. **Bright, P.** *Windows Phone 7 Series"Mango" for developers.* [tiešsaistes] – [atsauce 28.04.2011.]. Pieejams: http://arstechnica.com/microsoft/news/2011/04/windows-phone-7-mango-one-heck-of-an-upgrade.ars.
31. *Animating the player.* [tiešsaistes] – [atsauce 30.04.2011.]. Pieejams: http://create.msdn.com/en-US/education/tutorial/2dgame/animating_the_player.
32. *XNA and the Garbage Collector.* [tiešsaistes] – [atsauce 30.04.2011.]. Pieejams: http://channel9.msdn.com/Forums/TechOff/574844-XNA-and-the-Garbage-Collector.

PIELIKUMI

1. pielikums

Vienībtesti pārklāšanas metodes pārbaudei

```
[TestClass]
public class UnitTest1
{
    public UnitTest1() {}
    private TestContext testContextInstance;

    ...

    [TestMethod]
    // Vienāda izmēra objekti nepārklājās
    public void TestMethod1()
    {
        Circle obj1 = new Circle();
        obj1.Position = new Vector2(100, 100);
        obj1.Radius = 10;

        Circle obj2 = new Circle();
        obj2.Position = new Vector2(200, 200);
        obj2.Radius = 10;

        bool expected = false;
        bool actual;
        actual = obj1.intersect(obj2);
        Assert.AreEqual(expected, actual);
    }

    [TestMethod]
    // Objekts ir citā objektā
    public void TestMethod2()
    {

        Circle obj1 = new Circle();
        obj1.Position = new Vector2(100, 100);
        obj1.Radius = 10;

        Circle obj2 = new Circle();
        obj2.Position = new Vector2(100, 100);
        obj2.Radius = 20;

        bool expected = true;
        bool actual;
        actual = obj1.intersect(obj2);
        Assert.AreEqual(expected, actual);
    }

    [TestMethod]
    // Objekti pārklājas minimāli
    public void TestMethod3()
    {
```

```
        Circle obj1 = new Circle();
        obj1.Position = new Vector2(-50, 50);
        obj1.Radius = 100;

        Circle obj2 = new Circle();
        obj2.Position = new Vector2(-50, 160);
        obj2.Radius = 10;

        bool expected = true;
        bool actual;
        actual = obj1.intersect(obj2);
        Assert.AreEqual(expected, actual);
    }

    ...
}
```

2. pielikums

Riņķa līnijas klases pirmkods

```
public class Circle
{
    public int Radius;
    public Vector2 Position;

    public bool intersect(Circle obj)
    {
       /*
        * 1. algoritms
        distanceVector = Position - obj.Position;
        return (distanceVector.Length() <= Radius + obj.Radius);
       */
       /*
        * 2. algoritms neizmantojot kvadrātsakni
       float a, dx, dy;
       dx = Position.X - obj.Position.X;
       dy = Position.Y - obj.Position.Y;
       a = (Radius + obj.Radius);
       return (a * a >= (dx * dx) + (dy * dy));
       */

        // 3. algoritms
        // pirmā objekta apvilktais kvadrāts
        float Ax1 = Position.X - Radius, Ax2 = Position.X + Radius;
        float Bx1 = obj.Position.X - obj.Radius, Bx2 = obj.Position.X + obj.Radius;

        // otrā objekta apvilktais kvadrāts
        float Ay1 = Position.Y – Radius, Ay2 = Position.Y + Radius;
        float By1 = obj.Position.Y - obj.Radius, = obj.Position.Y + obj.Radius;

        // pārbaudīt vai kvadrāti pārklājās
        if (
            ((Bx1 <= Ax1 && Bx2 >= Ax1) || (Bx1 <= Ax2 && Bx2 >= Ax2) ||
```

```
                (Bx1 >= Ax1 && Bx2 <= Ax2))
               &&
               ((By1 <= Ay1 && By2 >= Ay1) || (By1 <= Ay2 && By2 >= Ay2) ||
               (By1 >= Ay1 && By2 <= Ay2))
              )
            {
                // gadījumā, ja pārklājās, pielietot 2. algoritmu
                float a, dx, dy;
                dx = Position.X - obj.Position.X;
                dy = Position.Y - obj.Position.Y;
                a = (Radius + obj.Radius);
                return (a * a >= (dx * dx) + (dy * dy));

            }
            return false;
        }
    }
```

3. pielikums

Spēles cikla pirmkods

```
// nepieciešamās bibliotēkas pieslēgšana
using System;
using System.Text;
using System.Threading;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Input.Touch;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Devices;
using Microsoft.Phone.Shell;

class GameplayScreen : GameScreen
{
    ContentManager content;
    Bubble bubble;
    Background bgLayer1, bgLayer2;

    Texture2D enemyTexture;
    List<Enemy> enemies;

    Health health;

    TimeSpan previousHealthTime, healtSpawnTime;
    TimeSpan enemySpawnTime, enemyMinSpawnTime, previousSpawnTime;
    TimeSpan fireTime, previousFireTime, rumblerTime;

    int maxImmCnt = 4;
    int currnetImmCnt = 0;
    int prevSecTimeScore = -1;
```

```
int score;
int pointToSpeedUp = 30;
int updatedSpeedScore = 30;
bool gameOverFlag = false;

Random random;
Vector2 logoVelocity;
Texture2D projectileTexture, explosionTexture, heartTexture;
List<Animation> explosions;
List<Projectile> projectiles;

SoundEffect shootingSound, shootingHardSound, explosionSound;
VibrateController rumbler;
StringBuilder endMsg;
SpriteFont font;

// konstruktors, notiek objektu inicializācija
public GameplayScreen()
{

    // atslēgt gaidīšanas režīmu
    PhoneApplicationService.Current.UserIdleDetectionMode =
          IdleDetectionMode.Disabled;

    bubble = new Bubble();
    bgLayer1 = new Background();
    bgLayer2 = new Background();

    enemies = new List<Enemy>();
    previousSpawnTime = TimeSpan.Zero;
    enemySpawnTime = TimeSpan.FromSeconds(2f);
    enemyMinSpawnTime = TimeSpan.FromSeconds(0.2f);
    random = new Random();

    projectiles = new List<Projectile>();
    fireTime = TimeSpan.FromSeconds(.5f);

    explosions = new List<Animation>();
    score = 0;
    health = new Health();
    previousHealthTime = TimeSpan.Zero;
    healtSpawnTime = TimeSpan.FromSeconds(35f);
    rumblerTime = TimeSpan.FromSeconds(0.1f);
    rumbler = VibrateController.Default;
    endMsg = new StringBuilder();

    ...
}

// ielādes metode
public override void LoadContent()
{
    if (content == null)
        content = new ContentManager(ScreenManager.Game.Services, "Content");

    Animation bubbleAnimation = new Animation();
```

```
    Texture2D playerTexture = content.Load<Texture2D>("bubbleAnimation");
    bubbleAnimation.Initialize(playerTexture, Vector2.Zero, 72, 72, 2, 100,
          Color.White, 1f, true);
    bubble.Initialize(bubbleAnimation, new Vector2(
        ScreenManager.GraphicsDevice.Viewport.Width / 2,
        ScreenManager.GraphicsDevice.Viewport.Height / 2)
    );

    // Load the background
    bgLayer1.Initialize(content, "bg/bg1",
          ScreenManager.GraphicsDevice.Viewport.Height, 1);
    bgLayer2.Initialize(content, "bg/bg2",
          ScreenManager.GraphicsDevice.Viewport.Height, 2);

    enemyTexture        = content.Load<Texture2D>("enemyAnimation");
    projectileTexture   = content.Load<Texture2D>("projectile");
    heartTexture        = content.Load<Texture2D>("heart");
    explosionTexture    = content.Load<Texture2D>("explosion");
    explosionSound      = content.Load<SoundEffect>("sounds/pop");
    shootingSound       = content.Load<SoundEffect>("sounds/shooting");
    shootingHardSound   = content.Load<SoundEffect>("sounds/boom");
    font                = content.Load<SpriteFont>("scores");

    health.Initialize(content.Load<Texture2D>("health"));

    ScreenManager.Game.ResetElapsedTime();

    ...
}


public override void UnloadContent()
{
    content.Unload();
}

private void AddExplosion(Vector2 position, Color explosionColor)
{
    // atskaņot skaņu
    if (Options.sound) explosionSound.Play();

    Animation explosion = new Animation();
    explosion.Initialize(explosionTexture, position, 104, 204, 9, 25, explosionColor,
          1f, false);
    explosions.Add(explosion);
}

private void AddProjectile(Vector2 position, Vector2 positionTo, bool unstoppable)
{
    Projectile projectile = new Projectile();
    projectile.Initialize(ScreenManager.GraphicsDevice.Viewport,
          projectileTexture, position, positionTo, unstoppable);
    projectiles.Add(projectile);
    if (Options.sound)
    {
        if (unstoppable)
        {
```

```
            shootingHardSound.Play();
        }
        else
        {
            shootingSound.Play();
        }
    }
}

private void AddHealth(GameTime gameTime)
{
    if (!health.Active && gameTime.TotalGameTime - previousHealthTime >
          healtSpawnTime)
    {
        health.Active = true;
        health.setPosition(new Vector2(random.Next(health.Radius,
          ScreenManager.GraphicsDevice.Viewport.Width - health.Radius),
          -health.Radius));
        previousHealthTime = gameTime.TotalGameTime;
    }
}

// pievienot jaunu ienaidnieku
private void AddEnemy()
{
    Animation enemyAnimation = new Animation();
    float scale = 1 - (random.Next(1, 7) - 4) / 10.0f;
    enemyAnimation.Initialize(enemyTexture, Vector2.Zero, enemyTexture.Height,
          enemyTexture.Height, 2, 30, Color.White, scale, true);
    Vector2 position = new Vector2(random.Next(-enemyTexture.Width / 2,
          ScreenManager.GraphicsDevice.Viewport.Width + enemyTexture.Width / 2),
           -enemyTexture.Height);

    Enemy enemy = new Enemy();
    bool imm = false;
    if (currnetImmCnt < maxImmCnt)
    {
        imm = random.Next(0, 10) < 3;
        if (imm)
        {
            currnetImmCnt++;
        }
    }
    enemy.Initialize(enemyAnimation, position, imm);
    enemies.Add(enemy);
}

private void UpdateExplosions(GameTime gameTime)
{
    for (int i = explosions.Count - 1; i >= 0; i--)
    {
        explosions[i].Update(gameTime);
        if (explosions[i].Active == false)
        {
            explosions.RemoveAt(i);
        }
    }
```

```
}

private void UpdateEnemies(GameTime gameTime)
{
    // pievienot jaunu, pēc noteiktā laika
    if (gameTime.TotalGameTime - previousSpawnTime > enemySpawnTime)
    {
        previousSpawnTime = gameTime.TotalGameTime;
        AddEnemy();
    }

    // atjaunot ienaidniekus
    for (int i = enemies.Count - 1; i >= 0; i--)
    {
        enemies[i].Update(gameTime, ScreenManager.GraphicsDevice.Viewport.Height);

        if (enemies[i].Active == false)
        {
            // If not active and health <= 0
            if (enemies[i].Health <= 0)
            {
                // Add an explosion
                AddExplosion(enemies[i].Position,
                        enemies[i].Immortal ? Color.Black : Color.White);
                if (!gameOverFlag) score += enemies[i].Value;
            }
            // kontrolēt maksimālo melno ienaidnkieku skaitu
            if (enemies[i].Immortal)
            {
                currnetImmCnt--;
            }
            enemies.RemoveAt(i);

        }
    }
}

// spēles atjaunošanas metode
public override void Update(GameTime gameTime, bool otherScreenHasFocus,
                            bool coveredByOtherScreen)
{
    base.Update(gameTime, otherScreenHasFocus, coveredByOtherScreen);

    // punktu pieskaitīšana par laiku
    if (gameTime.TotalGameTime.Seconds != prevSecTimeScore &&
          gameTime.TotalGameTime.Seconds % 2 == 0)
    {
        score += 1;
        prevSecTimeScore = gameTime.TotalGameTime.Seconds;
    }

    if (!IsActive) return;

    // atjaunot ienaidnieku skaitu
    if (score != 0 && updatedSpeedScore == score)
    {
        updatedSpeedScore = updatedSpeedScore + pointToSpeedUp;
```

```
            enemySpawnTime = enemySpawnTime - TimeSpan.FromSeconds(0.2f);
            if (enemySpawnTime < enemyMinSpawnTime) enemySpawnTime = enemyMinSpawnTime;
        }
        // atjaunot visus objektus
        updatePlayer(gameTime);
        bgLayer1.Update();
        bgLayer2.Update();

        UpdateEnemies(gameTime);
        UpdateCollision(gameTime);
        UpdateProjectiles();
        UpdateExplosions(gameTime);
        AddHealth(gameTime);
        health.Update(gameTime, ScreenManager.GraphicsDevice.Viewport.Height);
        ...
    }

    // atjaunot burbuli
    private void updatePlayer(GameTime gameTime)
    {
        // ACCELEROMETER
        Vector3 acceleration = Accelerometer.GetState().Acceleration;
        logoVelocity.X += acceleration.X;
        logoVelocity.Y += -acceleration.Y;

        // mainām burbuļa poziciju
        Vector2 newPosition = bubble.Position + logoVelocity;
        // pārbaudīt lai burbulis būtu vienmēr uz ekrāna
        Viewport viewport = ScreenManager.GraphicsDevice.Viewport;
        if (newPosition.X < bubble.Radius)
        {
            newPosition.X = bubble.Radius;
            logoVelocity.X = 0;
        }
        else if (newPosition.X > viewport.Width - bubble.Radius)
        {
            newPosition.X = viewport.Width - bubble.Radius;
            logoVelocity.X = 0;
        }

        if (newPosition.Y < bubble.Radius)
        {
            newPosition.Y = bubble.Radius;
            logoVelocity.Y = 0;
        }
        else if (newPosition.Y > viewport.Height - bubble.Radius)
        {
            newPosition.Y = viewport.Height - bubble.Radius;
            logoVelocity.Y = 0;
        }
        bubble.Position = newPosition;
        // pārtraukt spēli ja burbulim nav dzīves
        if (bubble.Health <= 0 && !gameOverFlag)
        {
            gameOverFlag = true;
            endGame(Translations.valueForKey("game_over"));
        }
```

```
    bubble.Update(gameTime);
}

// atjaunot lodes
private void UpdateProjectiles()
{
    for (int i = projectiles.Count - 1; i >= 0; i--)
    {
        projectiles[i].Update();
        if (projectiles[i].Active == false)
        {
            // nodzēst neaktīvo
            projectiles.RemoveAt(i);
        }
    }
}
...

// pārbaudīt visas kolizijas
private void UpdateCollision(GameTime gameTime)
{
    // burbuļa un ienaidnieku kolizijas
    bool bubble_hit = false;
    for (int i = 0; i < enemies.Count; i++)
    {
        if (bubble.intersect(enemies[i]))
        {
            bubble.Health -= enemies[i].Damage;
            enemies[i].Health = 0;
            bubble_hit = true;

            // vibrācija
            if (Options.vibration) rumbler.Start(rumblerTime);

            if (bubble.Health <= 0)
                bubble.Active = false;
        }
    }

    // lodes un ienaidnieku kolizijas
    for (int i = 0; i < projectiles.Count; i++)
    {
        for (int j = 0; j < enemies.Count; j++)
        {
            if (!enemies[j].Immortal && projectiles[i].intersect(enemies[j]))
            {
                if (projectiles[i].Unstoppable)
                {
                    enemies[j].Active = false;
                    enemies[j].Health -= enemies[j].Health;
                }
                else
                {
                    enemies[j].Health -= projectiles[i].Damage;
                    enemies[j].color = Color.White;
                    projectiles[i].Active = false;
                }
```

```
            }
        }
    }


    // pārbaudīt burbuļa un dzīves koliziju
    bool health_hit = false;
    if (health.Active && bubble.intersect(health))
    {
        health_hit = true;
        if (bubble.Health + health.AddsHealth > bubble.HealthLimit)
        {
            if (!gameOverFlag) score += health.Value;
        }
        else
        {
            bubble.Health += health.AddsHealth;
        }
        health.Active = false;
    }
    ...

    // mainīt krāsu kad notiek kolizija
    if (bubble_hit)
    {
        bgLayer2.color = Color.Black;
        bubble.PlayerAnimation.color = Color.Red;
    }
    else if (health_hit)
    {
        bgLayer2.color = Color.White;
        bubble.PlayerAnimation.color = Color.Green;
    }
    else
    {
        bgLayer2.color = Color.White;
        bubble.PlayerAnimation.color = Color.White;
    }

}

// notiek ievaddatu saņemšana un apstrāde
public override void HandleInput(InputState input, GameTime gameTime)
{

    if (input == null)
        throw new ArgumentNullException("input");

    // Look up inputs for the active player profile.
    int playerIndex = (int)ControllingPlayer.Value;

    KeyboardState keyboardState = input.CurrentKeyboardStates[playerIndex];
    GamePadState gamePadState = input.CurrentGamePadStates[playerIndex];

    // if the user pressed the back button, we return to the main menu
    PlayerIndex player;
    if (input.IsNewButtonPress(Buttons.Back, ControllingPlayer, out player))
```

```
    {
        endGame(Translations.valueForKey("game_exit"));
    }
    else
    {
        // lodes pievienošana
        foreach (GestureSample gesture in input.Gestures) {
            if (gesture.GestureType == GestureType.Tap ||
                 gesture.GestureType == GestureType.Hold)
            {
                if (gameTime.TotalGameTime - previousFireTime > fireTime)
                {
                    previousFireTime = gameTime.TotalGameTime;
                    AddProjectile(bubble.Position, gesture.Position,
                        gesture.GestureType == GestureType.Hold);
                }
            }
        }
    }

}

// spēle ir pabeigta, paradīt tastatūru
private void endGame(String headerMsg)
{
    endMsg.Remove(0, endMsg.Length);
    endMsg.Append(headerMsg);
    endMsg.Append("\n");
    endMsg.Append(Translations.valueForKey("your_result"));
    endMsg.Append(": ");
    endMsg.Append(score.ToString());

    if (!Guide.IsVisible)
            Guide.BeginShowKeyboardInput(
                    PlayerIndex.One,
                    endMsg.ToString(),
                    Translations.valueForKey("input_name"),
                    "",
                    new AsyncCallback(OnEndShowKeyboardInput),
                    null);
}


private void OnEndShowKeyboardInput(IAsyncResult result)
{
    String value = Guide.EndShowKeyboardInput(result);
    if (gameOverFlag == true || value != null)
    {
        if (!String.IsNullOrEmpty(value))
        {
            // gadījumā ja tiek ievadīts spēlētāja vārds, saglabājas rezultātu
            PlayerScores playerStat = new PlayerScores();
            playerStat.Name = value;
            playerStat.Scores = score;
            TopScores.add(playerStat);
        }
        // parejam uz galvēno izvēlni
```

```
            LoadingScreen.Load(ScreenManager, false, ControllingPlayer,
              new BackgroundScreen(), new MainMenuScreen());
        }
    }

    // Zīmēšanas metode
    public override void Draw(GameTime gameTime)
    {
        ScreenManager.GraphicsDevice.Clear(ClearOptions.Target, Color.CornflowerBlue,
              0, 0);
        SpriteBatch spriteBatch = ScreenManager.SpriteBatch;
        spriteBatch.Begin();
        // background
        bgLayer1.Draw(spriteBatch);
        bgLayer2.Draw(spriteBatch);

        // izvadīt ienaidniekus
        for (int i = 0; i < enemies.Count; i++) enemies[i].Draw(spriteBatch);
        health.Draw(spriteBatch);
        // lodes
        for (int i = 0; i < projectiles.Count; i++) projectiles[i].Draw(spriteBatch);
        // animāciju
        for (int i = 0; i < explosions.Count; i++) explosions[i].Draw(spriteBatch);
        ...
        bubble.Draw(spriteBatch);

        // izvadīt spēles punktus
        spriteBatch.DrawString(font, String.Format("{0:0000}", score),
              new Vector2(ScreenManager.GraphicsDevice.Viewport.TitleSafeArea.X + 10,
              ScreenManager.GraphicsDevice.Viewport.TitleSafeArea.Y), Color.White);

        // dzīvības līmeni
        for (int i = 0; i < bubble.Health / 10; i++)
        {
            spriteBatch.Draw(heartTexture,
              new Vector2(ScreenManager.GraphicsDevice.Viewport.Width - ((i + 1) *
                    heartTexture.Width + 10 * (i + 1)), 10), Color.White);
        }
        spriteBatch.End();
        // If the game is transitioning on or off, fade it out to black.
        if (TransitionPosition > 0)
              ScreenManager.FadeBackBufferToBlack(1f - TransitionAlpha);

    }
}
```

DOKUMENTĀRĀ LAPA

Bakalaura darbs „Spēles izstrāde Windows Phone 7 mobilajiem telefoniem” izstrādāts LU Datorikas fakultātē.

Ar savu parakstu apliecinu, ka pētījums veikts patstāvīgi, izmantoti tikai tajā norādītie informācijas avoti un iesniegtā darba elektroniskā kopija atbilst izdrukai.

Autors: Dmitrijs Ivanovs _____

Rekomendēju darbu aizstāvēšanai
Vadītājs: Dr. dat. Uldis Straujums _____

Recenzents: Mg. dat. Ilvars Mizniks _____

Darbs iesniegts Datorikas fakultātē 30.05.2011.
Metodiķe: Ārija Sproģe _____

Darbs aizstāvēts bakalaura gala pārbaudījuma komisijas sēdē
_____ prot. Nr. _____, vērtējums _____

Komisijas sekretārs: Dr. dat. Uldis Straujums _____