Vyšší odborná škola a Střední průmyslová škola elektrotechnická Olomouc,
Božetěchova 3

PRAKTICKÁ ZKOUŠKA Z ODBORNÝCH PŘEDMĚTŮ

Programování aplikace pro Google Android

2013

Jiří Vaculík

Prohlašuji, že jsem praktickou zkoušku vypracoval samostatně a všechny prameny jsem uvedl v seznamu použité literatury.

……

jméno a příjmení studenta

Chtěl bych vyslovit poděkování panu Ing. Marku Nožkovi za odborné konzultace a poskytnuté informace.

……

jméno a příjmení studenta

Prohlašuji, že nemám námitek proti půjčování nebo zveřejňování mé práce nebo její části se souhlasem školy.

……

jméno a příjmení studenta

OBSAH

Úvod

Mobilní operační systém Android od společnosti Google je na trhu již od roku 2009. Původně byl určen pro nasazení v chytrých telefonech, dnes ho však najdeme v mnoha dalších typech zařízení, jako jsou tablety, multimediální přehrávače, chytré televize, netbooky nebo úsporné minipočítače o velikosti flash disku.

V oblasti chytrých telefonů měl v roce 2012 68% podíl na trhu, což z něj činí nejpoužívanější platformu v tomto segmentu, která stále roste. Každým dnem je aktivován víc než 1 milion nových chytrých telefonů a tabletů s Androidem dohromady. Do těchto zařízení je každý měsíc instalováno více než 1,5 miliardy aplikací z obchodu Google Play [1]. S takovou poptávkou po aplikacích roste také poptávka po programátorech.

Mezi konkurencí tolika aplikací v Google Play už není jednoduché najít aplikaci či hru, která zde chybí. Pro programování jsem si vybral hru TetriNET mimo jiné také proto, že na platformě Android zatím není zastoupena (pokud nepočítám jiné Tetris klony). Hlavním cílem této práce je vytvořit herního klienta TetriNET, který bude dobře vypadat a bude uzpůsoben pro ovládání na dotykových obrazovkách mobilních telefonů a tabletů. Mým osobním cílem je dále také publikace této hry v obchodě Google Play a její monetizace.

Tato práce se v první kapitole zabývá obecnou problematikou programování aplikace pro Google Android, hrou TetriNET a síťového protokolu, který používá pro komunikaci serveru a klienta.

V druhé kapitole práce se budu zabývat realizací konkrétní aplikace, kterou jsem nazval Mobile TetriNET. Popíšu zde celý vývojový cyklus od návrhu grafického rozhraní přes samotné programování, kompilaci aplikace, její ladění a testování na konkrétních zařízeních.

Práci završuje třetí kapitola, kde se budu věnovat celému procesu publikace na Google Play a zvážím různé možnosti monetizace vhodné pro hru Mobile TetriNET.

1. Teoretický úvod

1.1. Programování pro platformu Android

Aplikace pro platformu Android jsou standardně vyvíjeny v jazyce Java za pomocí Android SDK [2]. Možné jsou však i jiné cesty.

1.1.1. Android SDK

Android Software Development Kit (SDK) je oficiální balík nástrojů pro vývoj aplikací na platformě Android. Tento balík obsahuje ladící software (debugger), knihovny, emulátor mobilního zařízení založený na QEMU, dokumentaci, ukázkové zdrojové kódy a návody [3].

V současné době jsou podporovány platformy Linux, Mac OS X 10.5.8 nebo novější a Windows XP nebo novější. Oficiálně podporované vývojové prostředí (IDE) je Eclipse s nainstalovaným modulem Android Development Tools (ADT). Dále je nutné, aby byl na pracovní stanici nainstalován nejnovější Java Development Kit (JDK) a v případě použití jiného IDE než Eclipse i Apache Ant [4].

Drtivá většina aplikací je vyvíjena právě tímto způsobem v jazyce Java se standartním rozhraním Android API, včetně většiny systémových aplikací. Vývojáři mají k dispozici sadu standartních widgetů pro tvorbu grafického uživatelského rozhraní (GUI).

1.1.2. Android NDK

Android Native Development Kit (NDK) umožňuje programátorům pro vývoj aplikací využívat knihovny napsané v jazyce C/C++. Tyto knihovny lze pomocí NDK zkompilovat do nativního kódu platforem ARM, MIPS a x86. Nativní třídy pak lze volat přes standartní Android API rozhraní. Knihovny mohou být zkompilovány na normálním PC se standartním kompilátorem, např. GCC [5].

Kompletní aplikace mohou být zkompilovány a nainstalovány přes standartní nástroje Android SDK. Na rozdíl od vyvíjení Java aplikací v prostředí Eclipse neprobíhá kompilace a instalace aplikace automaticky, ale je nutno využívat nástrojů v příkazové řádce a spouštět je ručně postupně pro kompilaci, instalaci a ladění.

Android NDK se nejčastěji používá k vývoji her a jiných aplikací náročných na výkon. Výhodou oproti Javě je mnohem větší rychlost, nevýhodou pak trochu složitější vývoj.

1.1.3. Python a knihovna Kivy

Open source knihovna Kivy pro jazyk Python slouží k vývoji aplikací s moderním dotykovým rozhraním pro velké množství platforem. Stejný kód můžeme spustit na počítači (Linux, Windows, Mac OS X), zařízení s Androidem (telefon, tablet) nebo zařízení s Apple iOS (iPhone, iPad). Části knihovny kritické pro výkon jsou implementovány v jazyce C [6].

V případě vývoje pro platformu Android využíváme Android SDK i NDK. Nejprve je nutno zkompilovat interpret jazyka Python přes NDK a následně vytvořit jednoduchou Java aplikaci, která spouští Python kód. Tyto dva úkony je nutno provést ručně přes příkazovou řádku pomocí skriptů dodaných od vývojářů projektu Python for Android. Vývoj pro Android je v tomto případě podporován pouze na platformách Linux a Mac OS X, pro Windows chybí potřebné nástroje.

V této práci je použita právě knihovna Kivy a jazyk Python. Výhodou této knihovny je její otevřenost, množství podporovaných platforem a jednoduchost. Nevýhodou je vyšší hardwarová náročnost oproti aplikacím vyvíjených čistě v C/C++.

1.2. Princip hry TetriNET

TetriNET je online hra typu Tetris, kterou může zároveň hrát až 6 hráčů. Cílem hry je vyřadit všechny soupeře a zůstat jako jediný hráč s nezaplněným herním polem.

Každý hráč skládá zeď z bloků padajících po obrazovce, každý blok se skládá ze 4 kostek a má svou pevně danou barvu. Existuje 7 typů bloků, které vyobrazují písmena I, J, L, O, S, T a Z (viz tabulka 1) [7].

Typ bloku	Značení v protokolu TetriNET	Vyobrazení
I	1	
O	2	
J	3	
L	4	
Z	5	
S	6	
T	7	

Tabulka 1 – Přehled herních bloků

Kompletně zaplněný řádek bez mezer zmizí a zbývající kostičky nad ním spadnou o jeden řádek níže. Současně je možné smazat maximálně 4 řádky pomocí bloku I, takový tah se nazývá tetris.

Potom, co hráč zaplní určitý počet řádků (zpravidla jeden), se na jeho poli některá kostička promění ve speciální. Při vyplnění řádku se speciální kostkou se tato kostka přidá na začátek hráčova inventáře. V případě vyplnění více řádků se do pole přidá odpovídající počet speciálních kostek.

Hráč může speciální kostky kdykoliv ze svého inventáře poslat na pole kteréhokoli hráče (včetně sebe). Ve hře TetriNET existuje celkem 9 typů speciálních kostek, viz tabulka č. 2 [8].

Značení		**Jméno**	**Ikona**	**Popis**
číselné	**písemné**			
1	a	add line		Přidá hráči jeden náhodně vyplněný řádek na konec pole
2	c	clear line		Vymaže poslední řádek z pole, kostičky nad ním spadnou
3	n	nuke field		Kompletně vymaže celé hrací pole
4	r	random clear		Náhodně vybere 10 buněk v herním poli a vymaže je
5	s	switch field		Vymění hrací pole mezi dvěma hráči
6	b	clear specials		Přemění speciální kostky na obyčejné (barvu náhodně vybere)
7	g	gravity		Všechny kostky spadnou dolů, vyplní mezery
8	q	quake field		Všechny řádky se náhodně posunou o 0 až 3 sloupce doleva nebo doprava
9	o	block bomb		Vymaže kostky typu block bomb a všechny okolní kostky vystřelí náhodně do prázdného místa v poli

Tabulka 2 – Přehled speciálních kostek a jejich značení

Pokud hráč vyplní více než 2 řádky, pak se všem protihráčům přidá na konec pole odpovídající počet nevyplněných řádků (pokud tuto vlastnost server podporuje). Viz tabulka 3 [9].

Počet vyplněných řádků	Pořet řádků přidaných ostatním
2	1
3	2
4	4

Tabulka 3 – přidávání řádků protihráčům

Součástí hry TetriNET je také herní chat, ve kterém spolu hráči mohou komunikovat pomocí krátkých zpráv. Do chatu posílá zprávy i server, např. pokud se nějaký hráč odpojí apod.

Na server se dále ukládají statistiky o počtu výher všech hráčů a týmů. Každý hráč si při připojení na server může napsat, v jakém je týmu. Týmy nemají ve hře žádné jiné využití, než ve výherních statistikách.

Každý hráč získá při připojení na server své identifikační číslo v rozsahu 1 až 6, a to podle pořadí v jakém se hráči připojili. Hráč s nejnižším číslem je vždy tzv. moderátor. Moderátor má právo spustit hru a pak ji kdykoli pozastavit nebo zrušit. Pokud se moderátor ze serveru odpojí, je tato hodnost předána dalšímu hráči s nejnižším číslem. V případě, že se původní moderátor do hry opět připojí, je mu tato hodnost navrácena.

1.3. Síťový protokol TetriNET

Protokol je v informatice konvence nebo standard, podle kterého probíhá elektronická komunikace a přenos dat mezi dvěma koncovými body. V nejjednodušší podobě protokol definuje pravidla řídící syntaxi, sémantiku a synchronizaci vzájemné komunikace [10].

Komunikace v TetriNET probíhá na TCP portu 31457. Existují 3 varianty protokolu TetriNET, konkrétně verze 1.13, 1.14 a varianta TetriFast. Tyto verze se liší pouze v detailech [11].

Komunikace je založena na textových zprávách, které si mezi sebou posílají server a klient. Každá taková zpráva je vždy ukončena ASCII znakem *FF* hexadecimálně (255 decimálně).

1.3.1. Zprávy SERVER – KLIENT

Přidělení ID hráči

Syntaxe:	`playernum <číslo>` **NEBO** `)#)(!@(*3 <číslo>`		
Popis:	První zpráva, kterou server poílá konkrétnímu klientovi. První variantu používá protokol verze 1.13 a 1.14, variantu druhou používá tetrifast.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo	celé číslo	1 - 6	Identifikační číslo hráče

Připojení nového hráče

Syntaxe:	`playerjoin <číslo> <jméno>`		
Popis:	Tato zpráva je poslána všem klientům, v okamžiku připojení nového hráče na server.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo	celé číslo	1 - 6	Identifikační číslo nového hráče
jméno	řetězec	-	Jméno nového hráče

Odpojení hráče

Syntaxe:	`playerleave <číslo>`		
Popis:	Zpráva je poslána všem klientům, v okamžiku odpojení nějakého hráče.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo	celé číslo	1 - 6	ID odpojeného hráče

Výběr týmu

Syntaxe:	`team <číslo> <název týmu>`		
Popis:	Zpráva zaslána všem klientům, kromě klienta, který tým měnil. Odeslána ihned po připojení hráče nebo kdykoli v průběhu hry, když hráč změní svůj tým.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo	celé číslo	1 - 6	ID hráče, který mění tým
název týmu	řetězec	-	Jméno týmu, může být vynecháno - bez týmu

Statistiky výher

Syntaxe:	`winlist <typ><jméno>;<skóre> ... <typ><jméno>;<skóre>`		
Popis:	Odesláno každému klientovi hned při připojení a vždy při ukončení hry. Obsahuje statistiky s hráči a týmy s nejvyšším počtem výher.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
typ	znak	t nebo p	Tým (t) nebo hráč (p)
jméno	řetězec	-	Jméno týmu nebo hráče
skóre	celé číslo	-	Počet výher týmu nebo hráče

Zpráva v herním chatu

Syntaxe:	`pline <číslo> <zpráva>`		
Popis:	Odesláno všem klientům, kromě odesílatele, vždy když někdo napíše do herního chatu.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo	celé číslo	1 - 6	ID odesílatele zprávy
zpráva	řetězec	-	Obsah zprávy

Nová hra

Syntaxe:	TetriNET 1.13:	`newgame <1> <2> … <11>`	
	TetriNET 1.14:	`newgame <1> <2> … <11> <seed>`	
	TetriFast:	`******* <1> <2> … <11>`	
Popis:	Tato zpráva je odeslána všem klientům v okamžiku, kdy moderátor odstartuje hru.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
1	celé číslo	≥ 0	Počet náhodně vyplněných řádků, kderé se objeví na konci pole při startu hry
2	celé číslo	≥ 1	Počáteční úroveň (čím vyšší úroveň, tím rychlejší padání kostek)
3	celé číslo	≥ 1	Počet řádků, které hráč musí vyplnit, aby postoupil na další úroveň
4	celé číslo	≥ 0	Počet úrovní, o který se hráč posune při vyplnění počtu řádků nutného k postupu do další úrovně
5	celé číslo	≥ 1	Počet řádků, které hráč musí vyplnit, aby se v poli objevily speciální kostky
6	celé číslo	≥ 0	Počet přidaných speciálních kostek při vyplnění počtu řádků nutného k přidání speciálních kostek
7	celé číslo	≥ 0	Kapacita inventáře pro speciální kostky. Pokud bude mít hráč zaplněný inventář, další speciální kostky se nepřidají, dokud nějakou speciální kostku nepoužije.
8	řetězec 100 celých čísel	1 - 7	Frekvenční tabulka výskytu různých bloků (každé číslo představuje 1%)
9	řetězec 100 celých čísel	1 - 9	Frekvenční tabulka výskytu různých speciálních kostek (každé číslo představuje 1%)
10	boolean	1 nebo 0	Pokud je 1, klient by měl zobrazit průměrnou úroveň všech hráčů. Pokud je 0, tato hodnota je skryta.
11	boolean	1 nebo 0	Pokud je 1, je povoleno přidávání řádků ostatním hráčům při vyplnění více než dvou.
seed	celé číslo	$0 - 2^{32}$	Počáteční hodnota (tzv. seed) pro generátor bloků. Podporuje verze TetriNET 1.14 - všem klientům budou padat stejné bloky.

Ve hře

Syntaxe:	`ingame`
Popis:	Informační zpráva o tom, že v daném kanálu serveru probíhá hra a připojený klient je jen pozorovatel.

Aktualizace úrovně

Syntaxe:	`lvl <číslo hráče> <úroveň>`		
Popis:	Posláno všem hráčům tehdy, když se změní úroveň některého hráče.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo hráče	celé číslo	1 - 6	Číslo hráče, jehož úroveň se mění.
úroveň	celé číslo	-	Nová úroveň

Použití speciální kostky

Syntaxe:	`sb <číslo hráče> <typ> <odesílatel>`		
Popis:	Posláno všem hráčům, když někdo odešle speciální kostku.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo hráče	celé číslo	0 - 6	Číslo udávající ID hráče, na kterého byla speciální kostka poslána. (0 značí všechny hráče)
typ	znak	a, b, c, g, n, o, q, r, s	Typ speciální kostky
odesílatel	celé číslo	0 - 6	Číslo udávající ID hráče, který speciální kostku odeslal. (0 značí server)

Přidávání řádků

Syntaxe:	`sb 0 cs<počet> <odesílatel>`		
Popis:	Pokud to server podporuje, při vyplnění několika řádků hráčem se ostatním přidá odpovídající počet řádků navíc. Je to pouze lehce modifikovaná varianta předchozí zprávy, číslo hráče je vždy 0, jelikož se řádky přidávají všem.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
počet	celé číslo	1, 2 nebo 4	Hodnota není vždy sejná jako počet vyplněných řádků.
odesílatel	celé číslo	0 - 6	Hráč, který vyplnil několik řádků současně. (0 značí přidání řádků serverem všem hráčům)

Prohra

Syntaxe:	`playerlost <číslo hráče>`		
Popis:	Posláno všem hráčům jako informace o tom, že některý hráč prohrál - zaplnil celé své pole		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo hráče	celé číslo	1 - 6	Číslo hráče, který prohrál.

Výhra			
Syntaxe:	`playerwon <číslo hráče>`		
Popis:	Posláno všem hráčům, když už všichni až na jednoho mají své pole plné.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo hráče	celé číslo	1 - 6	Číslo hráče, který vyhrál.

Pozastavení a návrat do hry			
Syntaxe:	`pause <stav>`		
Popis:	Odesláno všem hráčům vždy když je hra pozastavena nebo obnovena. V případě připojení hráče do probíhající hry tato zpráva automaticky následuje za zprávou *ingame* a informuje, zda je zrovna probíhající hra pozastavena.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
stav	boolean	0 nebo 1	1 značí pozastavení hry, 0 návrat do hry

Konec hry	
Syntaxe:	`endgame`
Popis:	Odesláno všem hráčům tehdy když je hra ukončena. Hra končí, pokud některý hráč vyhraje, anebo když moderátor hru předčasně ukončí.

Nepřipojeno			
Syntaxe:	`noconnecting <důvod>`		
Popis:	Odesláno klientovi, kterému server zamítnul připojení.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
důvod	řetězec	-	Chybová zpráva popisující důvod odmítnutí připojení na server.

Tabulka 4 – Zprávy SERVER – KLIENT [12]

1.3.2. Zprávy KLIENT – SERVER

Přihlášení na server			
Syntaxe:	`tetrisstart <jméno> <ver>` **NEBO** `tetrifaster <jméno> <ver>`		
Popis:	První zpráva, kterou klient odesílá serveru. Tato zpráva slouží k přihlášení na server. První verze platí pro protokol TetriNET 1.13 a 1.14, druhou používá TetriFast. Zpráva je zakódována, viz kapitola 2.2.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
jméno	řetězec	-	Jméno hráče
ver	řetězec	1.13 nebo 1.14	Verze TetriNET protokolu

Název týmu

Syntaxe:	`team <číslo> <název týmu>`		
Popis:	Zasláno serveru okamžitě po obdržení hráčského ID. Udává jméno týmu, za který hraje lokální hráč.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo	celé číslo	1 - 6	Identifikační číslo nového hráče
jméno	řetězec		Jméno nového hráče

Zpráva v herním chatu

Syntaxe:	`pline <číslo> <zpráva>`		
Popis:	Odesláno serveru, když lokální hráč odešle zprávu do herního chatu.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo	celé číslo	1 - 6	ID odesílatele zprávy
zpráva	řetězec		Obsah zprávy

Spuštění / ukončení hry

Syntaxe:	`startgame <stav> <číslo>`		
Popis:	Odesláno serveru moderátorem (hráčem s nejnižším ID) pro odstartování nebo ukončení hry.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
stav	boolean	1 nebo 0	1 odstartuje hru, 0 ukončí hru
číslo	celé číslo	1 - 6	ID lokálního hráče, musí být nejnižší ze všech hráčů

Aktualizace úrovně

Syntaxe:	`lvl <číslo> <úroveň>`		
Popis:	Odesláno v okamžiku, kdy lokální hráč dosáhne vyšší úrovně ve hře. To je tehdy, když hráč vyplní určitý počet řádků.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo	celé číslo	1 - 6	ID lokálního hráče.
úroveň	celé číslo	-	Nová úroveň.

Použití speciální kostky

Syntaxe:	`sb <číslo hráče> <typ> <odesílatel>`		
Popis:	Odesláno serveru, pokud lokální hráč někomu odeslal speciální kostku.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo hráče	celé číslo	1 - 6	Číslo udávající ID hráče, na kterého byla speciální kostka poslána.
typ	znak	a, b, c, g, n, o, q, r, s	Typ speciální kostky
odesílatel	celé číslo	1 - 6	Číslo udávající ID lokálního hráče, který speciální kostku odeslal.

Přidávání řádků			
Syntaxe:	`sb 0 cs<počet> <odesílatel>`		
Popis:	Pokud to server podporuje, při vyplnění několika řádků lokálním hráčem se ostatním přidá odpovídající počet řádků navíc. Je to pouze lehce modifikovaná varianta předchozí zprávy, číslo hráče je vždy 0, jelikož se řádky přidávají všem.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
počet	celé číslo	1, 2 nebo 4	Hodnota není vždy sejná jako počet vyplněných řádků.
odesílatel	celé číslo	1 - 6	ID lokálního hráče.
Prohra			
Syntaxe:	`playerlost <číslo hráče>`		
Popis:	Odesláno serveru v případě, že už má lokální hráč zaplněné pole a nemůže přidávat další bloky. Některé servery očekávají, že před odesláním této zprávy klient odešle aktualizaci s náhodně vyplněným polem.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
číslo hráče	celé číslo	1 - 6	ID lokálního hráče.
Pozastavení a návrat do hry			
Syntaxe:	`pause <stav> <číslo hráče>`		
Popis:	Odesláno všem hráčům vždy když je hra pozastavena nebo obnovena. V případě připojení hráče do probíhající hry tato zpráva automaticky následuje za zprávou *ingame* a informuje, zda je zrovna probíhající hra pozastavena.		
Argumenty	**Typ**	**Hodnoty**	**Význam**
stav	boolean	0 nebo 1	1 značí pozastavení hry, 0 návrat do hry
číslo hráče	celé číslo	1 - 6	ID lokálního hráče, musí být nejnižší ze všech hráčů

Tabulka 5 – Zprávy KLIENT – SERVER [13]

1.3.3. AKTUALIZACE HERNÍHO POLE

Aktualizace herního pole je společná zpráva pro server i klient. Klient odesílá serveru aktualizaci svého pole, vždy když dopadne herní blok nebo je přijata speciální kostka. Server klientům posílá aktualizace polí všech hráčů, tak aby mohl klient všechna pole zobrazit. Existují dva typy aktualizace – plná a částečná. Odesílaná zpráva vypadá v obou případech následovně [14]:

`f <číslo hráče> <řetězec pole>`

Argument `<číslo hráče>` je celé číslo, obsahuje ID hráče, jehož pole se má aktualizovat. Nabývá hodnot od 1 do 6.

Obsah argumentu `<řetězec pole>` závisí na typu aktualizace, viz dále.

Plná aktualizace je odeslána vždy při startu hry, dále už je odesílána pouze částečná. Řetězec v tomto případě obsahuje 264 znaků (12 x 22), kde znaky představují jednotlivé buňky herního pole v pořadí zleva doprava, shora dolů. Znaky mohou být buď celá čísla (0 je prázdná buňka, 1 – 5 barevné kostky) nebo malá písmena (speciální kostky – a, b, c, g, n, o, q, r, s). Řetězec může vypadat např. takto:

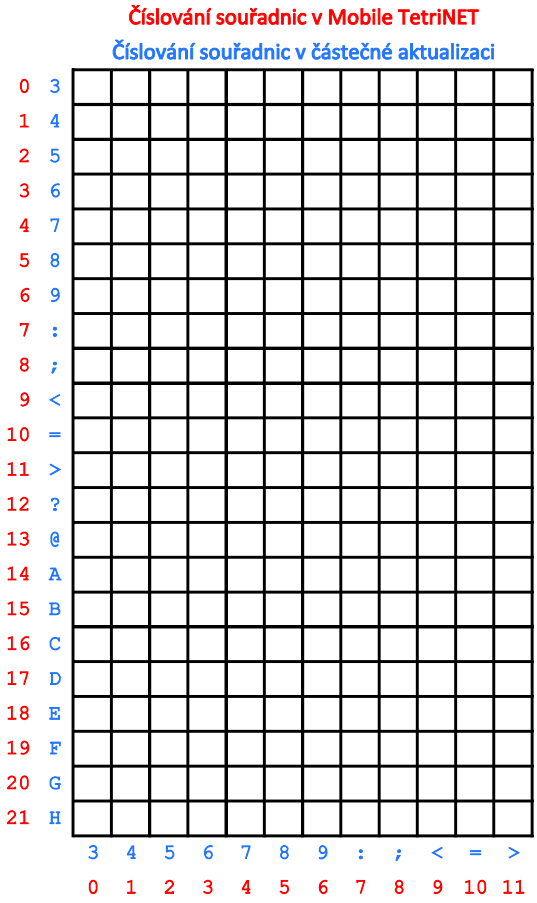
```
00000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000
0000000000000000000000510000000005551110100 0a55551101000555
5a5101022r525521010225222c22
```

Částečná aktualizace je sice složitější, ale neobsahuje tolik znaků. V řetězci se odesílají pouze ty buňky, které se oproti minulé aktualizaci změnily. První znak částečné aktualizace se v ASCII tabulce nachází od 21 do 2F hexadecimálně. Viz tabulka 6. Tento znak se v řetězci může vyskytovat vícekrát.

Znak	Pozice v ASCII [HEX / DEC]	Typ buňky	Obrázek
!	21 / 33	prázdná	
"	22 / 34	modrá	
#	23 / 35	žlutá	
$	24 / 36	zelená	
%	25 / 37	fialová	
&	26 / 38	červená	
'	27 / 39	add line	
(	28 / 40	clear line	
)	29 / 41	nuke field	
*	2A / 42	random clear	
+	2B / 43	switchfield	
,	2C / 44	clear specials	
-	2D / 45	gravity	
.	2E / 46	quakefield	
/	2F / 47	block bomb	

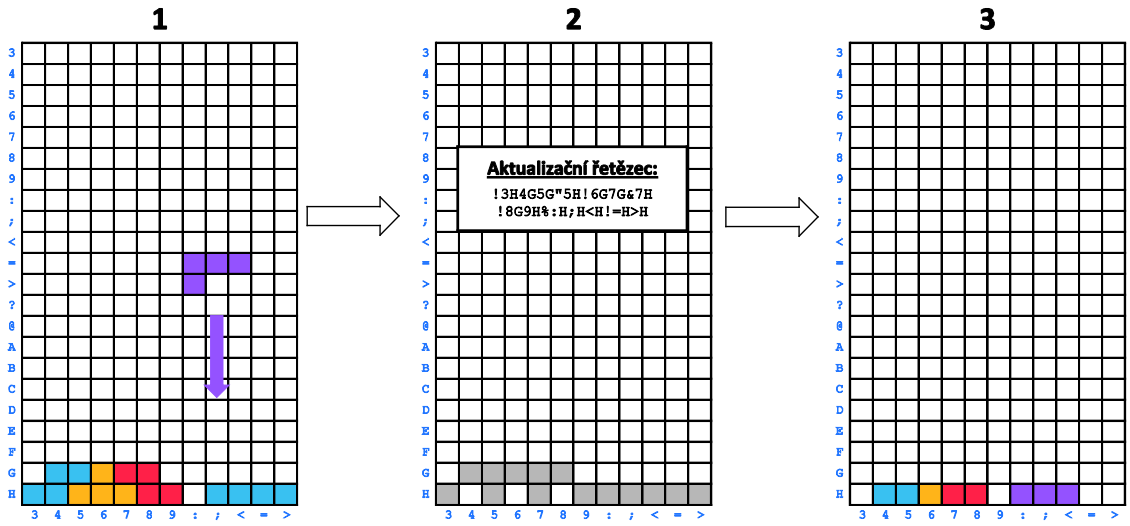
Tabulka 6 – Kódování částečné aktualizace pole

Další 2 znaky představují X, Y souřadnice konkrétní buňky. Pozice se číslují zleva doprava, shora dolů a začínají znakem 3 (33 HEX, 51 DEC). Vyšší pozice představují znaky s vyšším ASCII kódem. Viz obrázek 1 – 1.



Obrázek 1 – 1 Systém souřadnic v Mobile TetriNET a částečných aktualizacích

Na obrázku 2 – 1 v 1. kroku dopadne fialový blok, spodní řádek se tedy vyplní a horní spadne dolů. Ve 2. kroku jsou znázorněny rozdílové buňky, ze kterých sestavujeme aktualizační řetězec. 3. krok představuje výsledné zobrazení pole.



Obrázek 1 – 2 Sestavení řetězce pro částečnou aktualizaci

2. REALIZACE

2.1. OBJEKTOVĚ ORIENTOVANÉ PROGRAMOVÁNÍ

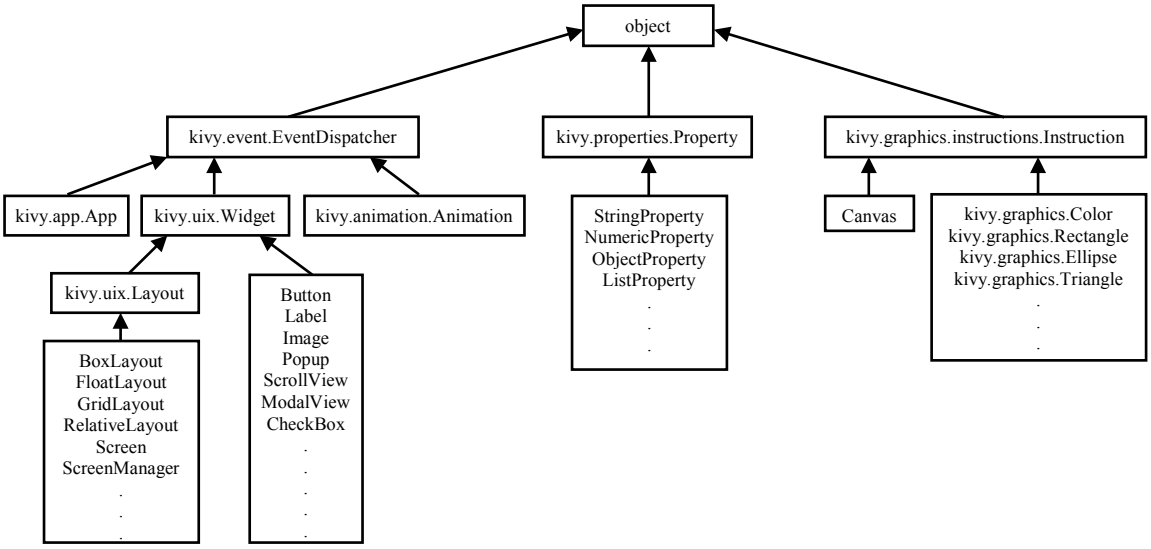
Objektově orientované programování (zkráceně OOP z anglického Object-oriented programming) je metodika vývoje softwaru, založená na následujících myšlenkách [15]:

- *Objekty* – jednotlivé prvky modelované reality (jak data, tak související funkčnost) jsou v programu seskupeny do entit, nazývaných objekty. Objekty si pamatují svůj stav a navenek poskytují operace (přístupné jako metody pro volání).
- *Abstrakce* – programátor, potažmo program, který vytváří, může abstrahovat od některých detailů práce jednotlivých objektů. Každý objekt pracuje jako černá skříňka, která dokáže provádět určené činnosti a komunikovat s okolím, aniž by vyžadovala znalost způsobu, kterým vnitřně pracuje.
- *Zapouzdření* – zaručuje, že objekt nemůže přímo přistupovat k „vnitřnostem“ jiných objektů, což by mohlo vést k nekonzistenci. Každý objekt navenek zpřístupňuje *rozhraní*, pomocí kterého (a nijak jinak) se s objektem pracuje.
- *Skládání* – objekt může obsahovat jiné objekty.
- *Delegování* – objekt může využívat služeb jiných objektů tak, že je požádá o provedení operace.
- *Dědičnost* – objekty jsou organizovány stromovým způsobem, kdy objekty nějakého druhu mohou *dědit* z jiného druhu objektů, čímž přebírají jejich schopnosti, ke kterým pouze přidávají svoje vlastní rozšíření. Tato myšlenka se obvykle implementuje pomocí rozdělení objektů do *tříd*, přičemž každý objekt je instancí nějaké třídy.
- *Polymorfismus* – odkazovaný objekt se chová podle toho, jaké třídy je instancí. Pokud několik objektů poskytuje stejné rozhraní, pracuje se s nimi stejným způsobem, ale jejich konkrétní chování se liší podle implementace.

Jazyk Python je tzv. víceparadigmatický, to znamená, že umožňuje při psaní programů využívat nejen objektově orientované paradigma, ale i procedurální nebo funkcionální. V aplikaci Mobile TetriNET však využívám výhradně objektově orientovaný způsob, jelikož jedině tak mohu plně využít možnosti, které nabízí použitá knihovna Kivy. Díky OOP je zdrojový kód lépe pochopitelný, přehlednější a kratší.

2.1.1. Použité třídy Kivy

Knihovna Kivy nabízí velmi bohaté a přehledné API (aplikační rozhraní). V následujícím přehledu naleznete většinu tříd, které jsou použity v aplikaci Mobile TetriNET.

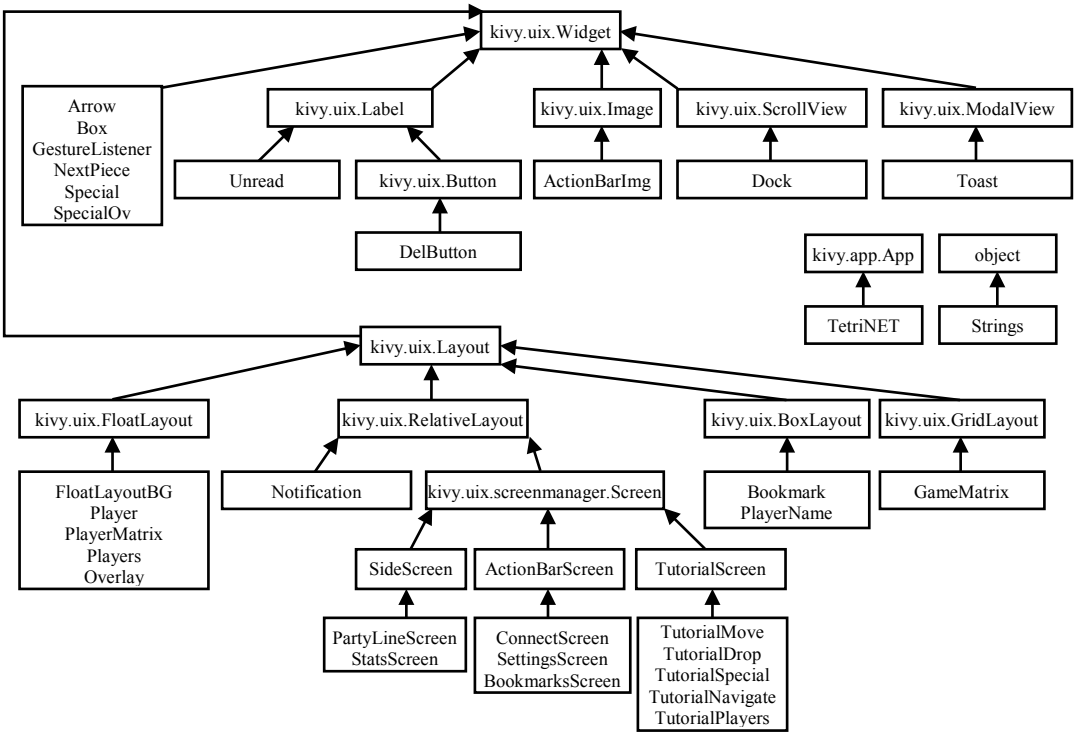


Obrázek 2 – 1 – Přehled použitých tříd knihovny Kivy

- `kivy.app.App` – hlavní třída, která řídí běh celé aplikace
- `kivy.uix.Widget` – tato a veškeré další odvozené třídy představují objekty uživatelského rozhraní s určitou grafickou reprezentací a funkcemi
- `kivy.uix.Layout` – slouží ke skládání více „widgetů“ do bloků dle určitých pravidel definovaných v konkrétním „layoutu“
- `kivy.properties.Property` – tato a další třídy od ní odvozené slouží pro ukládání různých dat. Na rozdíl od klasických datových typů v Pythonu se zde při změně hodnoty vytvářejí události, na které mohou být mapovány funkce.
- `kivy.graphics.instructions.Instruction` – grafické instrukce pro kreslení

2.1.2. VLASTNÍ TŘÍDY

V následujícím přehledu naleznete všechny mnou napsané třídy použité v této aplikaci. Téměř všechny jsou rozšířením některé ze základních tříd Kivy z předchozí kapitoly.



Obrázek 2 – 2 – Přehled všech vlastních tříd

- `TetriNET` – hlavní třída, která řídí celou aplikaci. Obsahuje hlavní smyčku a metody pro zpracování a odesílání zpráv protokolu TetriNET.
- `Box` – třída reprezentující jednu kostičku v herním poli
- `GameMatrix` – reprezentuje herní pole lokálního hráče, obsahuje 12 x 22 objektů `Box` rozmístěných v mřížce. Tato třída také obsluhuje herní logiku – generuje herní bloky, řídí jejich padání a ovládání, generuje řetězec pro aktualizaci herního pole, atd. Více v kapitole 2.3.
- `Overlay` – vysouvací lišta pro zobrazení ostatních hráčů. Obsahuje instance tříd `Player`, `PlayerName`, `Players`, `PlayerMatrix`, `Arrow`.
- `SideScreen`, `ActionBarScreen`, `TutorialScreen` – grafické rozhraní se skládá z několika typů obrazovek, více v kapitole 2.4.
- `Strings` – jediná vlastní třída, která není odvozena z knihovny Kivy. Slouží pro import jazyků (čeština, angličtina).

2.2. KOMUNIKACE SE SÍTÍ

Komunikaci se síťovým rozhraním na nejnižší úrovni zajišťuje knihovna Twisted, která je oficiálně podporována tvůrci knihovny Kivy. V Kivy je předpřipraveno rozhraní, díky kterému může Twisted přímo přistupovat do hlavní smyčky Kivy aplikace.

Z knihovny Twisted dědí dvě mnou vytvořené třídy, viz následující ukázka kódu:

```
class EchoClient(twisted.internet.protocol.Protocol):
    def connectionMade(self):
        self.factory.app.on_connection(self.transport)
    def dataReceived(self, data):
        self.factory.app.print_message(data)

class EchoFactory(twisted.internet.protocol.ClientFactory):
    protocol = EchoClient
    def __init__(self, app):
        self.app = app
    def clientConnectionLost(self, conn, reason):
        …
    def clientConnectionFailed(self, conn, reason):
        …
```

Všechna data proudí přes třídu `EchoClient`, která má v sobě instanci `EchoFactory`. Třída `EchoFactory` má přístup do hlavní aplikační třídy `TetriNET`, v níž může volat různé metody v případě rozpadu spojení.

Spojení se serverem navazuje metoda `TetriNET.connect_to_server()`, která vytvoří instanci `EchoFactory` a naváže prvotní spojení s vybraným serverem. Po zdárném navázání spojení se pomocí události zavolá metoda `TetriNET.on_connection(connection)`, která serveru ihned odešle zprávu pro přihlášení klienta.

Přihlašovací zpráva protokolu TetriNET se šifruje speciálním algoritmem [16]:

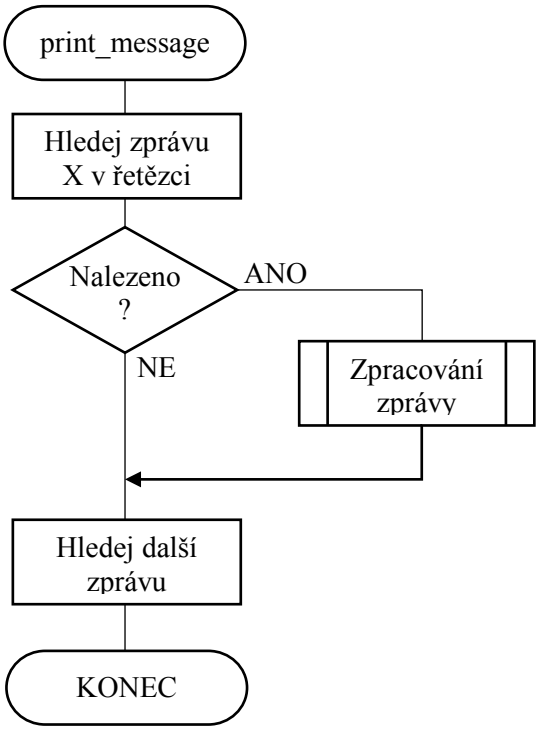
```
def loginEncode(self):
      msg = "tetrisstart {1} {2}".format(self.nickname, self.version)
      h = [int(y) for y in self.serverIP.split(".")]
      h = str(54*h[0] + 41*h[1] + 29*h[2] + 17*h[3])
      dec = randint(0, 255)
      encodedStr = "{0:02X}".format(dec)
      for i in range(len(msg)):
            dec = ((dec + ord(msg[i]))%255) ^ ord(h[i % len(h)])
            encodedStr += "{0:02X}".format(dec)
      return encodedStr.upper()
```

Např. pokud se hráč *Pihvi* přihlašuje s protokolem verze 1.13 a klasickým herním módem (`tetrisstart Pihvi 1.13`) na server s IP adresou 127.0.0.1, při `randint(0,255) = 128` bude zakódovaná přihlašovací zpráva vypadat takto:

```
80C210B3134A85CF71E46FD4C124B529AA227A9CFF0702
```

Pozn.: Mód TetriFast se přihlašuje se zprávou: `tetrifaster Pihvi 1.13`

Příjem a odesílání zpráv protokolu TetriNET mají na starost metody `TetriNET.print_message(data)` a `TetriNET.send_message(msg)`. Jelikož řetězec, který přijde ze serveru, může obsahovat několik zpráv (vždy oddělených znakem *FF* HEX), testuji tento řetězec pro každou zprávu zvlášť. Viz vývojový diagram:



Obrázek 2 – 3 Vývojový diagram zpracování zpráv ze serveru

Při nalezení zprávy v řetězci se vykoná posloupnost operací specifická pro danou zprávu. V této posloupnosti se většinou volají metody „widgetů“, které pak provedou příslušné akce.

V případě odesílání zpráv serveru z klienta si „widgety“ samy volají metodu `TetriNET.send_message(msg)`.

2.3. HERNÍ LOGIKA

Herní logiku kompletně obstarává třída `GameMatrix`, je zde řízeno náhodné generování bloků, padání, testování vyplnění řádků, generování speciálních kostek a jejich aplikace na lokálního hráče, aktualizace herního pole, pohyb bloků, atd.

Při startu hry je vygenerován seznam pěti herních bloků, které se budou postupně zobrazovat v herním poli. Při každém spadení herního bloku je na konec seznamu vygenerován další blok. Jelikož modifikace TetriFast podporuje generování stejných bloků napříč klienty, musel jsem do aplikace implementovat stejný náhodný generátor, jaký mají i ostatní TetriNET klienty [17].

Funkce vypadá následovně:

```
def generate(self):
    a = 0x08088405
    c = 1
    M = 2**32
    self.seed = (a*self.seed + c) % M
    block = int(self.seed*100./M)
    self.seed = (a*self.seed + c) % M
    orientation = int(self.seed*4./M)
return (block,orientation)
```

Tato funkce vrací neměnitelný seznam s číslem v rozsahu 0 – 100 reprezentující blok a dalším číslem 0 – 3 reprezentující počáteční otočení bloku. Číslo bloku je pozice znaku ve frekvenční tabulce posílané serverem při startu hry.

V případě, že hrajeme mód TetriFast, server sám pošle počáteční hodnotu (tzv. seed) pro náhodný generátor. V případě, že hrajeme klasický mód, seed generuje standardní generátor náhodných čísel v Pythonu. Seed je 32 bitové celé číslo.

Posílání speciálních kostek na lokální pole je realizováno příslušnými metodami třídy `GameMatrix`, které jsou volány z třídy `TetriNET` v okamžiku, kdy přijde zpráva ze serveru.

2.4. Grafické rozhraní

Základní myšlenkou mobilního dotykového grafického rozhraní je přepínání mezi několika „obrazovkami“. Např. při spuštění hry se zobrazí hlavní menu, uživatel klikne třeba na tlačítko „Připojit“, nyní hlavní menu zmizí a zobrazí se místo něj obrazovka s dialogovými okny pro přihlášení na server. Pokud se chce uživatel vrátit do hlavního menu, stiskne na svém zařízení systémové tlačítko zpět a obrazovka se změní na hlavní menu. Takto to funguje na mnoha mobilních operačních systémech včetně OS Android, v některých případech (konkrétně Apple iOS) systémové tlačítko chybí a programátor musí do GUI zahrnout své vlastní navigační tlačítko.

Dalším důležitým aspektem je velikost ovládacích prvků. Jelikož dotykové ovládání prstem je relativně nepřesné, musí mít všechny aktivní grafické prvky dostatečnou velikost. Velikost je spojena i s hustotou zobrazení (PPI). Dnešní moderní zařízení mají hustotu zobrazení okolo 300 až 400 ppi (nejlepší displeje mají dnes rozlišení 1920 x 1080 na úhlopříčce okolo 5“), proto je nutné veškerou grafiku zvětšovat dle tohoto parametru.

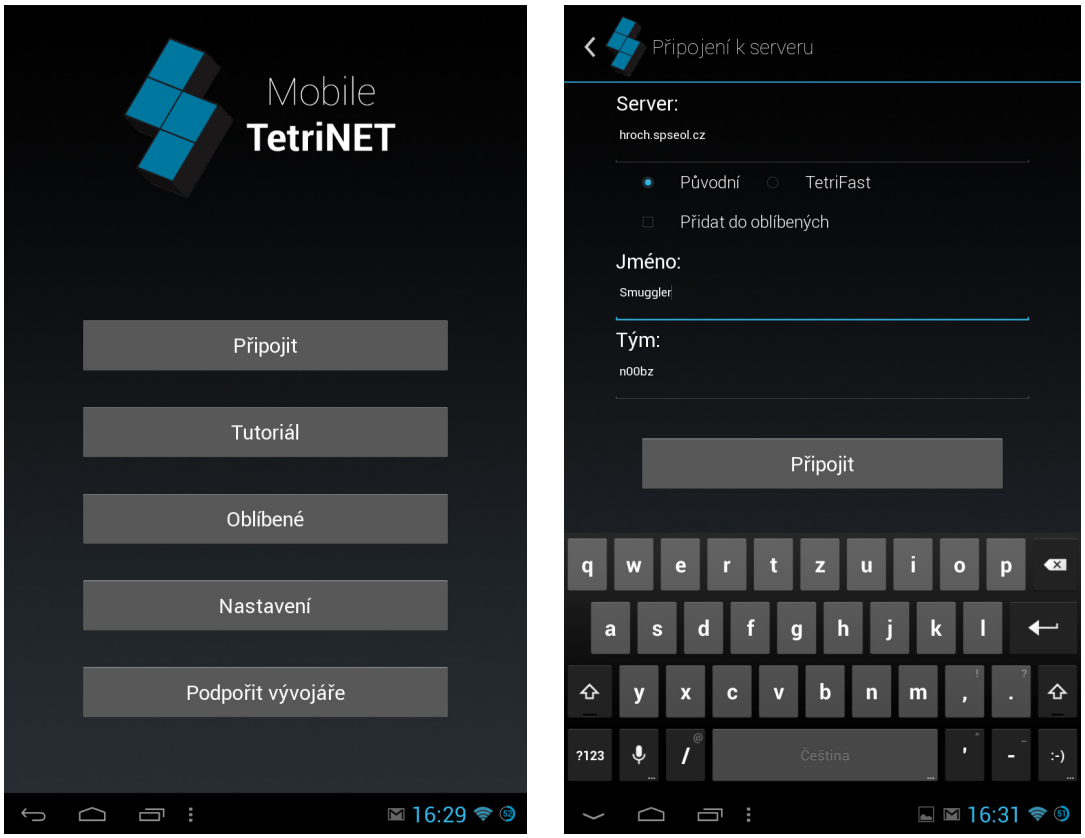
V aplikacích pro Android verze 4 a vyšší je doporučeno se při tvorbě GUI řídit tzv. Android Design Guidelines [18]. Společnost Google nabízí volně k dispozici (bez

licenčních omezení) pro tvorbu aplikací několik ikon a grafických prvků pro použití v aplikacích. Některé ikony jsou použity i v Mobile TetriNET.

Grafické rozhraní může být spravováno přímo v Pythonu, avšak je lepší použít speciální jazyk „Kivy Language“, který je svým principem podobný CSS. Kód Kivy Language je uložen v souborech s koncovkou *.kv. Syntaxe jazyka je velice jednoduchá a podobná Pythonu, jazyk umožňuje spravovat vzhled pro jednotlivé „widgety“ včetně umísťování několika „widgetů“ do sebe, mapování funkcí k událostem, přístup k proměnným Pythonu, atd. Příklad:

```
<MujWidget>:
      vlastnost1: self.atributMeTridy
      vlastnost2: root.atributNadrazenehoWidgetu
      Button:
           text: "Tlačítko"
           on_press: root.stiskTlacitka()
```

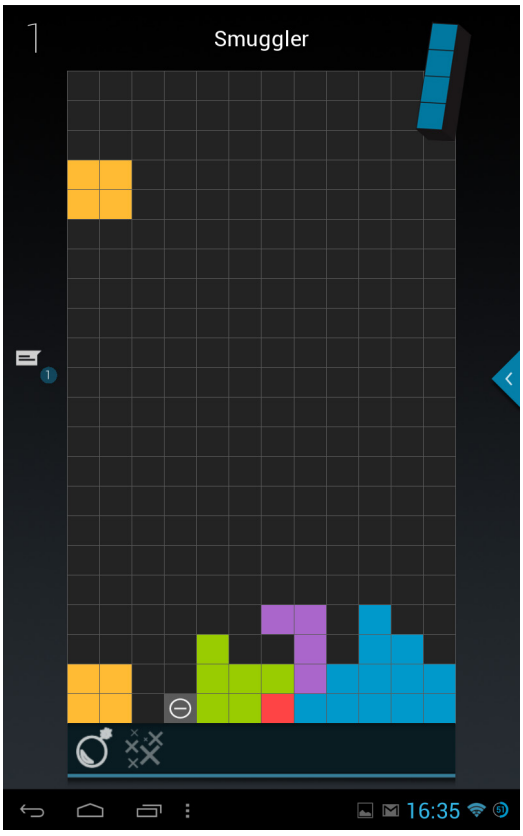
O přepínání mezi jednotlivými obrazovkami aplikace se stará třída `kivy.uix.screenmanager.ScreenManager`, která obsahuje všechny obrazovky a řídí přepínání mezi nimi. Jednotlivé obrazovky mohou být třídy `kivy.uix.screenmanager.Screen` nebo od ní odvozené. V praxi má každá obrazovka svou vlastní odvozenou třídu s vlastním vzhledem.



Obrázek 2 – 4 Ukázka vzhledu aplikace. Vlevo hlavní menu, vpravo připojení k serveru.

Naprosto každý objekt třídy `Widget` může přijímat události `on_touch_down`, `on_touch_move` a `on_touch_up`, které se vytvářejí při dotyku prstu, pohybu a zvednutí z obrazovky. Na těchto událostech je postaveno veškeré dotykové ovládání.

Například u herního pole lokálního hráče. Pokud se uživatel dotkne obrazovky a začne s prstem pohybovat, metoda `on_touch_move` třídy `GameMatrix` zaznamenává pozici prstu a pokud se prst pohne o určitý počet pixelů, pak se posune i aktuálně padající blok. Pokud se uživatel pouze dotkne pole a prst ihned zvedne, pak v závislosti na pozici dotyku se padající blok otočí doprava nebo doleva (toto je řízeno v metodě `on_touch_up`).



Obrázek 2 – 5 Vzhled herní obrazovky

2.5. Vytvoření instalačního balíčku

Všechny aplikace pro OS Android se instalují pomocí instalačního balíčku s koncovkou APK. Jedná se v podstatě o trochu upravený ZIP archiv, APK balíček bez problému otevřete např. v programu WinRAR.

Vytváření balíčku je u aplikacích vytvářených v prostředí Eclipse v jazyce Java extrémně jednoduché, stačí jedno kliknutí. U aplikacích Kivy je to trochu složitější, avšak díky podpoře tvůrců této knihovny stále poměrně jednoduché.

Pro zabalení Kivy aplikace do instalačního balíčku APK potřebujeme [19]:

- Počítač nebo virtuální stroj s OS Linux (nejlépe Ubuntu)
- Nainstalovanou Javu (JDK – Java Development Kit)
- Nainstalovaný Python 2.7 a modul Jinja2
- Nástroj Apache Ant
- Android SDK a NDK

Dále potřebujeme zprovoznit sadu nástrojů Python for Android dle návodu [20] a následně stáhnout potřebné soubory prostřednictvím GIT repozitářů:

```
$ git clone git://github.com/kivy/python-for-android
```

Nyní už můžeme zkompilovat vlastní distribuci jazyku Python s moduly, které používáme v aplikaci, pomocí skriptu `distribute.sh` (součást Python for Anroid):

```
$ ./distribute.sh -m 'android kivy twisted sdl' final
```

Jazyk Python stačí zkompilovat pouze jednou (pokud v průběhu vývoje nebudeme přidávat další moduly) a v případě vydání další verze aplikace ji už stačí znovu zabalit do instalačního balíčku. Pro vytvoření balíčku APK slouží skript `build.py` opět dodávaný v projektu Python for Android. Tento skript se nachází ve složce distribuce vytvořené pomocí `distribute.sh`.

```
$ ./build.py --private "/home/smuggler/Mobile
TetriNET/" --package com.smuggler.tetrinetdebug --name
"Mobile TetriNET" --compile-pyo --icon
/home/smuggler/icon.png --presplash
/home/smuggler/presplash2.jpg --install-location
"internalOnly" --permission INTERNET --permission VIBRATE
--version 0.9.3.2 --orientation portrait debug installd
```

- *--private* – tento parametr zahrnuje soubory, které se v zařízení nainstalují do interní paměti dostupné pouze pro mou aplikaci
- *--package* – udává jméno Java balíčku aplikace
- *--name* – zobrazované jméno aplikace
- *--compile-pyo* – zdrojové kódy předkompiluje do formátu PYO (Python Optimized)
- *--icon* – cesta k obrázku s ikonou aplikace
- *--presplash* – cesta k obrázku, který se zobrazí při načítání aplikace

- --*install-location* – specifikuje, kam se aplikace bude instalovat. V mém případě pouze do interní paměti zařízení, lze nastavit i externí paměť (SD kartu)
- --*permission* – udává Android oprávnění aplikace, Mobile TetriNET má oprávnění k plnému přístupu do internetu a pro vibrace.
- --*version* – číslo verze
- --*orientation* – otočení aplikace *portrait* nebo *landscape* (na ležato)
- *debug* – může být *debug* nebo *release* (viz kapitola 3.1)
- *installd* – po zabalení se aplikace nainstaluje na připojené zařízení přes USB, v případě parametru *release* nutno použít *installr*

Po spuštění už skript sám vytvoří instalační balíček, který nainstalujeme na zařízení s OS Android.

Pro ladění aplikace na zařízení už musíme použít nástroj z Android SDK zvaný „logcat“, který slouží pro výpis logu. Spustíme jej následujícím příkazem (zařízení musí být připojeno v USB nebo na stejné síti přes WiFi):

```
adb logcat
```

3. PUBLIKACE

Hlavním zdrojem aplikací pro OS Android je obchod Google Play Store. Existují i další obchody, např. Amazon nebo Samsung Apps. Aplikaci lze šířit i pouhým zveřejněním instalačního APK balíčku. V této kapitole se budu zabývat pouze publikací aplikace v Google Play.

3.1. PODEPSÁNÍ APLIKACE

Před nahráním do obchodu Google Play je nutné aplikaci zabalit s parametrem `release` a připojit svůj elektronický podpis. Elektronický podpis nemusí být nijak ověřený a informace v něm vyplněné jsou pro publikaci irelevantní, důležitý je pouze jeho kontrolní součet. Tento podpis už musí být pro každou vydanou aplikaci stejný, nelze jej změnit. [21]

Svůj privátní klíč jsem vygeneroval v programu *keytool*, dodávaném v JDK. Privátní klíč můžeme generovat např. s následujícími parametry:

```
$ keytool -genkey -v -keystore my-release-key.keystore
-alias alias_name -keyalg RSA -keysize 2048 -validity 10000
```

Dalším krokem je vytvoření instalačního balíčku v režimu `release`:

```
$ build.py release
```

Konečně můžeme aplikaci podepsat pomocí nástroje jarsigner, např. takto:

```
$ jarsigner -verbose -sigalg MD5withRSA -digestalg SHA1
-keystore my-release-key.keystore MobileTetriNET.apk
alias_name
```

Po spuštění tohoto příkazu se program zeptá na heslo k privátnímu klíči a aplikaci podepíše. Jako další krok je dobré ověřit, zda byla aplikace správně podepsána:

```
$ jarsigner -verify my_signed.apk
```

Posledním krokem je optimalizace balíčku, která všem nekomprimovaným datům přidá speciální hlavičku. Výsledkem je nižší spotřeba RAM při spouštění aplikace:

```
$ zipalign -v 4 MobileTetriNET-unaligned.apk
MobileTetriNET.apk
```

Nyní už je vše připraveno k nahrání aplikace do Google Play.

3.2. Obchod Google Play

Potom, co je aplikace připravena a podepsána, můžeme aplikaci nahrát do obchodu Google Play.

Pro publikaci v Google Play je potřeba založit si tzv. „vývojářský účet“, který nám zpřístupní administrační nástroje pro správu aplikací. Založení účtu je zpoplatněno jednorázovým poplatkem 25 USD (cca 480 Kč) [22]. Pro porovnání s platformou iOS, přístup do Apple iTunes stojí 99 USD ročně. V Google Play už bez dalších poplatků lze na jeden účet publikovat libovolný počet aplikací.

Administrační rozhraní je velmi jednoduché. Stačí do něj nahrát podepsaný instalační balíček, systém sám dle informací v balíčku určí podporovaná zařízení. Seznam podporovaných zařízení lze pak dále manuálně redukovat a upravovat.

V systému musíme vytvořit tzv. „listing“ aplikace, který obsahuje název, informace a popis aplikace, ikonu, doplňující grafiku, alespoň 3 snímky obrazovky, atd. Popis aplikace můžeme sami přeložit do jiných jazyků nebo zvolit strojový překlad.

V dalším kroku vybereme seznam zemí, kde bude naše aplikace dostupná, a konečně můžeme aplikaci publikovat. Zveřejnění aplikace trvá řádově několik hodin, tento čas není přesně stanoven.

Po zveřejnění můžeme sledovat bohaté statistiky o počtu instalací, odinstalací aplikace, verzích systému Android, atd.

V případě, že chceme aplikaci aktualizovat a vydat novou verzi, stačí do administrace nahrát nový instalační balíček. Aktualizace trvá opět několik hodin (ale méně než prvotní zveřejnění aplikace). Potom, v závislosti na nastavení konkrétního zařízení, se buď aplikace aktualizuje automaticky, nebo manuálně.

Obrázek 3 – 1 QR kód s odkazem na stránku aplikace v Google Play

3.3. Možnosti monetizace

V zásadě existují tři hlavní možnosti monetizace aplikací, které se ale dají různě kombinovat.

Jednou z možností je umístit do aplikace reklamu, většinou se jedná o obrázkový reklamní „banner“. Reklama v aplikacích se vyskytuje velmi často a tyto aplikace bývají většinou dostupné zdarma. Programátor si může vybrat různé reklamní systémy, oficiální a pravděpodobně nejčastější je Google AdMob. Výdělek závisí na počtu zobrazení reklamy (tedy i počtu uživatelů aplikace) a také na počtu kliknutní. Reklama se samozřejmě nezobrazuje bez přístupu k internetu a existují aplikace, které zobrazení reklamy blokují.

Další možnost je nabízet aplikaci ke koupi v obchodě Google Play. Takto šířené aplikace už většinou reklamu neobsahují. Cena aplikací se pohybuje od 15 Kč (nejjednodušší hry a aplikace) až po několik set korun (kancelářské balíky). Pirátství zde představuje velký problém. Takto zakoupené aplikace lze do 15 minut vrátit, je to problém pokud má uživatel na svém telefonu tzv. „root oprávnění“. Na telefonu s „root oprávněními“ lze zakoupenou aplikaci kamkoliv zkopírovat a pak ji vrátit a získat peníze zpět. Tyto zkopírované aplikace se pak šíří na internetových diskusích, kde už ji i bez „root oprávnění“ lze bez problému nainstalovat. Společnost Google si z ceny aplikace bere 30% podíl jako svůj zisk.

Třetí možností je tzv. „freemium“ model neboli „in-app purchase“, což znamená nakupování přímo v aplikaci. Nejčastěji se používá u her. Tyto hry jsou šířeny zdarma, ale umožňují nakupování virtuálních předmětů, které usnadňují postup hrou. Nákupy jsou propojené s Google Play a opět lze s „root oprávněními“ jednoduše nakupovat zdarma. Google si i z těchto nákupů bere 30% podíl.

V aplikaci Mobile TetriNET uvažuji o nasazení reklamy, kterou doplním možností přispět libovolnou částkou přes online peněženku PayPal.

3.4. Hardwarové nároky

Nároky na hardware jsou trochu vyšší, než je zvykem u podobných her. Je to pravděpodobně dáno tím, že Python je skriptovací jazyk, a tudíž je mnohem pomalejší než Java nebo C++. HW náročnost určitě také ovlivňuje to, že každá kostička na herním poli je novou instancí třídy Widget. Na jednom zaplněném herním poli je tedy 264 objektů typu Widget a herních polí je ve hře celkem 7 (pole lokálního hráče + pole všech 6 hráčů v panelu vpravo). Při zaplnění všech hráčských polí je to celkem 1848 objektů, což při testování na stolním PC nečinilo žádný problém, avšak na starším

mobilním zařízení už byl pokles výkonu znát. Současná moderní zařízení už s výkonem nebudou mít žádný problém.

Mobile TetriNET je nejlépe optimalizován pro tablet Asus Nexus 7 a mobilní telefon Samsung Nexus S. Všechna zařízení s označením „Nexus“ jsou referenční zařízení určená pro vývojáře, speciálně vyrobena pro společnost Google od jejích výrobních partnerů. Všechna tato zařízení se vyznačují tzv. „čistým Androidem“, tj. bez grafických nástaveb, a vždy jsou to první zařízení, které dostávají aktualizaci na novou verzi OS Android.

Aplikace byla dále testována na těchto zařízeních: Sony Xperia J, Sony Xperia Ray, Samsung Galaxy SII, Samsung Galaxy Ace II, HTC Desire HD, HTC Inspire 4G, HTC One X, Asus Transformer TF300TG a Prestigio Multipad PMP5570C DUO.

Na některých zařízeních se vyskytovaly chyby specifické pro dané zařízení, ve většině případů byly tyto chyby vyřešeny optimalizací kódu. Jedna chyba však přetrvává, konkrétně nefunkčnost standartní klávesnice dodávané na zařízeních značky Samsung (kromě Nexus S). Tato chyba je známá, a vyskytuje se ve všech aplikacích vytvořených v knihovně Kivy. Tento problém se dá vyřešit instalací jiné klávesnice nebo použitím vlastní klávesnice Kivy.

Na základě testovaných zařízení doporučuji minimální HW konfiguraci:

- 512 MB operační paměti RAM (nejlépe 1 GB)
- 1 GHz procesor ARM Cortex-A8 (nejlépe 2 a více jádrový Cortex-A9)
- Rozlišení displeje 800 x 480 (nejlépe 960 x 540 nebo 1280 x 800)
- Grafický čip s podporou Open GLES 2.0 je nutností
- Minimálně 30 MB volného místa v interní paměti

Jedná se o minimální testované specifikace, kromě podpory Open GLES 2.0 a dostatku místa v interní paměti mohou být nároky o něco nižší.

ZÁVĚR

V této práci byla vytvořena aplikace Mobile TetriNET, což je implementace původní hry TetriNET pro zařízení s operačním systémem Google Android. Aplikace umožňuje připojení k hernímu TetriNET serveru přes mobilní 3G síť nebo WiFi, kde lze tuto hru (s libovolným klientem) hrát s až pěti dalšími hráči.

Hra je přeložena do českého a anglického jazyka, jazyk lze zvolit při prvním spuštění hry a později případně změnit v nastavení. Hra dále umožňuje ukládat herní servery do záložek pro pozdější snadné připojení.

Aplikace má přehledné a intuitivní grafické uživatelské rozhraní optimalizované pro dotykové obrazovky různých velikostí. Design byl vytvořen dle tzv. „Android Design Guidelines“ za pomocí softwarového balíku Adobe Creative Suite 6.

Aplikace je naprogramována v jazyce Python 2.7 s knihovnou Kivy 1.6.1, grafické styly jsou napsány v jazyce Kivy Language. Aplikaci jsem programoval ve vývojovém prostředí Spyder v operačním systému Ubuntu Linux.

Vzhledem k použitým technologiím má tato aplikace o něco vyšší hardwarové nároky, než ostatní podobné hry. Je to dáno náročností jazyka Python a velkým množstvím objektů uložených v paměti. Tento problém se dá vyřešit lepší optimalizací kódu tak, aby se tolik objektů nevytvářelo.

Dalším problémem je nekompatibilita s klávesnicí dodávanou na zařízeních značky Samsung, která je zapříčiněna horší podporou konkrétní technologie v knihovně Kivy. Problém bude pravděpodobně vyřešen v některé novější verzi této knihovny, dočasně se dá vyřešit už nyní – instalací alternativní klávesnice nebo použitím virtuální klávesnice dodávané v Kivy.

Po odstranění těchto chyb mám v plánu přidat do aplikace reklamu a prezentovat ji na internetu za účelem přivýdělku. Aplikace pro mě bude mít využití i jako reference pro budoucí zaměstnání.

SEZNAM POUŽITÉ LITERATURY A STUDIJNÍCH MATERIÁLŮ

1) http://developer.android.com/about/index.html - Úvod k OS Android, 12. 3. 2013

2) http://developer.android.com/tools/index.html – Android Developer Tools úvod, 12. 3. 2013

3) http://developer.android.com/tools/help/index.html – Nástroje Android SDK, 12. 3. 2013

4) http://developer.android.com/sdk/index.html – Android SDK – požadavky na systém, 12. 3. 2013

5) http://en.wikipedia.org/wiki/Android_software_development - Vývoj aplikací pro Android na Wikipedii, 11. 3. 2013

6) http://kivy.org/#home – O knihovně Kivy, 12. 3. 2013

7) https://github.com/xale/iTetrinet/wiki/block-types – Dokumentace klienta iTetrinet – typy bloků, 12. 3. 2013

8) https://github.com/xale/iTetrinet/wiki/special-blocks – Dokumentace iTetrinet – Seznam speciálních kostek, 13. 3. 2013

9) https://github.com/xale/iTetrinet/wiki/classic-style-adds – Dokumentace iTetrinet – Klasické přidávání řádků, 13. 3. 2013

10) http://cs.wikipedia.org/wiki/Síťový_protokol – Wikipedia otevřená encyklopedie, 14. 3. 2013

11) http://jetrix.sourceforge.net/dev-guide.php#section2-3 – Dokumentace serveru Jetrix, 16. 3. 2013

12) https://github.com/xale/iTetrinet/wiki/tetrinet-protocol%3A-server-to-client-messages – Dokumentace iTetrinet – komunikace SERVER – KLIENT, 16. 3. 2013

13) https://github.com/xale/iTetrinet/wiki/tetrinet-protocol%3A-client-to-server-messages – Dokumentace iTetrinet – komunikace KLIENT – SERVER, 16. 3. 2013

14) https://github.com/xale/iTetrinet/wiki/field-updates – Dokumentace iTetrinet – Aktualizace pole, 16. 3. 2013

15) http://cs.wikipedia.org/wiki/Objektov%C4%9B_orientovan%C3%A9_programov%C3%A1n%C3%AD – Objektově orientované programování na Wikipedii, 20. 3. 2013

16) http://www.javafacts.com/jts/protocol.html – TetriNET protokol, 20. 3. 2013

17) http://jetrix.sourceforge.net/dev-guide.php#section2-1 – Dokumentace serveru Jetrix, náhodný generátor, 22. 3. 2013

18) http://developer.android.com/design/index.html – Návrh designu pro Android aplikace, 23. 3. 2013

19) http://kivy.org/docs/guide/packaging-android.html#packaging-android – Dokumentace Kivy, tvorba APK balíčku, 28. 3. 2013

20) http://python-for-android.readthedocs.org/en/latest/prerequisites/ – Python for Android, 28. 3. 2013

21) http://developer.android.com/tools/publishing/app-signing.html – Android Developers, podepsání aplikace, 25. 3. 2013

22) http://developer.android.com/distribute/googleplay/publish/register.html – Android Developers, registrace do Google Play, 25. 3. 2013

SEZNAM OBRÁZKŮ, GRAFŮ A TABULEK