Tetris – Dokumentace

*Zápočtový program, Tetris, je zábavný program, který lze využít buď pro zabití času hraním legendárních her, nebo pro edukativní cíle. Program obsahuje klasickou hru tetris, s typickým bodovým systémem a pravidly, která jsou drtivé většině lidí na planetě Zemi známy. Mimo to program obsahuje 2 různé umělé inteligence, které (za vhodných podmínek) dokážou hrát tetris i hodiny. Nejenom herní pamětníci zde najdou další bonusovou hru, kterou nejspíš ve svých mladých létech pařili na gameboyích.*

# Programátorská sekce

## Úvod

Program je napsán v jazyce C# .NET 5.0 a jedná se o Windows Forms App. Program obsahuje 2 různé hry, které se ovládají pomocí klávesnice a 2 umělé inteligence, které dokážou hrát tetris.

Tento program obsahuje nemalé množství různých programovacích prvků, o kterých se zvídavý zájemce může dozvědět na přednáškách NPRG031. Mezi zmíněné prvky patří například: abstraktní třída s abstraktními metodami, statická třída, dědičnost, přetěžování funkcí, předávání parametrů pomocí reference, přepisování funkci ve třídách potomstva, přetypování proměnných, dosazování synovského objektu do mateřské proměnné, různé datové struktury (2 typy front, zásobník), globální proměnné, pestrá škála klíčových slov pro modifikaci přístupu k metodám či atributům. Formulářová aplikace pracuje s událostmi jako je Button\_Click, PictureBox\_Paint, Timer\_Tick. Za zmínku stojí i možnost si, při hraní nebo pozorování AI, pustit tetrisovou hudbu.

## Část I – Hra Tetris

Kapitola 1 – hrací deska, místo pro radost i rozruch

Hrací deska pro tetris a obě umělé inteligence v daném projektu je reprezentována pomocí objektu třídy GameBoard, který je v hlavní třídě Form1.cs uložen pod názvem GameBoard gb. Při zapnutí (nebo i přepnutí) kteréhokoliv režimu se vždy zinicializuje nová hrací deska – gb = new GameBoard(); –. Objekt této třídy obsahuje dvojrozměrné pole public char[,] Board, které je v konstruktoru defaultně naplněno nulovým charem ‘\0‘, což je chápáno jako prázdné pole, místo, kam se může TetroBlock posunout nebo rotovat. Při hraní her uživatelem nebo v případě AI režimu se zaplňuje toto pole různými chary, které symbolizují jednotlivé barvy v konkrétním místě ve hře. Tyto informace jsou pak použity při zobrazování a vykreslování hrací desky a hracích figurek. Hrací deska má rozměry 10x20, ale zobrazováno je jenom dolních 18 řad. Horní 2 sice nejsou zobrazovány, ale i přesto může hráč (například při rotaci) do nich zavítat. Tato třída kromě hracího pole obsahuje celočíselné hodnoty level, lines a score. Nejpodstatnější z této trojice čísel je level, tato hodnota upravuje rychlost hry. Dochází ke zrychlení hry až do 10. levelu (kdy je rychlost hry 140ms), pak už je zrychlení nulové.

Třída GameBoard je mimo jiné používána pro generaci TetroBlocků. Generace není přímo náhodná. Existuje 7 druhů TetroBlocků, se kterými můžeme hrát tetris. Generaci si můžeme představit následovně. Do měšce vložíme všech 7 existujících TetroBlocků a postupně je po jednom vytahujeme. V okamžiku, kdy je měšec prázdny, navrátíme všechny TetroBlocky zpět a začínáme od znova. Tím pádem je záruka, že každých 13 generací (v nejhorším případě) dostanu chtěný TetroBlock. Tuto úlohu řeší statická metoda static public Shape GeneratePiece(), která ukládá potřebné informace do statických proměnných static int numOfPieces a static bool[] piecesDistribution. Návratová hodnota této funkce je Shape, což je abstraktní mateřská třída všech 7 různých TetroBlocků, proto můžeme jednoduše dosadit a přetypovat proměnnou Shape na potřebný objekt. Další podstatné metody této třídy jsou:

* public bool AddToBoard(Shape shp)
* public int[] FindFullLines()/static public int[] FindFullLines(ref char[,] deska)
* static public void MoveMap(ref char[,] deska, int[] lines)
* static public void ClearLines(ref GameBoard gb, int[] lines)

První zmíněná metoda vrátí true, pokud se nám podaří vložit aktivní hrací TetroBlock do hrací desky, tím pádem to patří ke stěžejním metodám, která nám prozradí, kdy je třeba hru ukončit, protože už není místo pro další TetroBlock. Jedná se o klasický cyklus přes pole int[,] Pozice, který otestuje, zda každá pozice v poli char[,] Board má nulový char jako hodnotu v potřebném místě. Druhá metoda patří k nejdůležitějším a je vyvolána následně po funkci AddToBoard. Tato metoda zkontroluje, jestli náhodou nedošlo k nějakému naplnění řádku a vrátí pole s čísly řádků, které je třeba vymazat. Při mazaní řádků dochází k úpravě hodnot level, score, lines, které jsou zobrazovány ve formuláři.

kapitola 2 – malujeme kostičky

Formulářová aplikace obsahuje celkově 20 prvků: 3x PictureBox, 5x Button, 5x Timer, 6x Label a 1x CheckBox. Všech 6 labelů složí pro zobrazování aktuálních údajů běžící hry. CheckBox se používá pro zapínání a vypínání Tetris Theme Song. Funkce každého tlačítka je zřejmá z textu, který je napsán na jeho tlačítku. Při přepnutí do kteréhokoliv režimu se nejprve vypnout všechny timery, vynulují se globální proměnné pro chod programu, vytvoří se nové objekty s „čistým štítem“, nastaví se správně hodnoty score a vygenerují se nové TetroBlocky, které jsou uloženy v proměnných Shape activePiece a Shape nextPiece. Jak už název napovídá, tak tyto proměnné definují figurku, se kterou hráč (popř. umělá inteligence) aktuálně hraje a manipuluje na hracím poli a se kterou bude hrát až aktivní figurka se vloží do desky. Figurka uchovává svoji barvu a svoje 4 pozice, na kterých se „virtuálně“ nachází v hracím poli. Je zde použito slovo virtuálně, protože dokud je figurka aktivní a je s ní možnost zacházet a manipulovat, tak není zaznamenána v hracím poli. K této změně dojde po zavolání metody třídy GameBoard: public bool AddToBoard(Shape shp).

Nachází se zde 5 timerů, 2 jsou uživatelské, pro 2 herní módy, 2 jsou pro 2 umělé inteligence a poslední je pro „střílení“ ve hře Wall Breaker, což je kromě pohybu doleva a doprava, jediný akční prvek. Zbývají 3 pictureboxy. Všechny hry a režimy se odehrávají v prvním z nich. 2. je čistě dekorační, pakliže obsahuje jenom tetris logo a ve 3. se zobrazuje následující TetroBlock, který je uložen v proměnné Shape nextPiece.

Tvorba a design jsou uskutečněny pomocí statické třídy Visual, která odpovídá za správné namalování hrací desky a figurek. Nutnost přemalovat hrací plochy zařídí Paint Event neboli událost, která je vyvolána pomocí funkce PictureBox[13].Invalidate(). Tato funkce na oba živé pictureboxy je volána při každém zmáčknutí každého tlačítka (kromě EXIT tlačítka) a ve všech událostech timer[12345]\_Tick. Příslušný Paint Event nejprve ověří, že hrací deska i aktivní figurka již byly inicializovány a následně zavolá metodu Visual.DrawGame(ref GameBoard gb, Shape shp, Graphics grafika, Pen tuzka). Tato funkce se skládá z dalších 2 funkcí pro zobrazení hrací desky a aktivní figurky. Pro obě funkce je stěžejní bratrská funkce void DrawRect(Graphics grafika, Pen tuzka, char color, int height, int width). Tato funkce namaluje jeden krásný čtvereček potřebné barvy a umístí ho podle maticových souřadnic height & width do správné polohy v herním poli.

Kapitola 3 – třída shape a jejich 7 dětí

Jak už bylo zmíněno, tak existuje zde abstraktní třída Shape, která obsahuje jak abstraktní metody nebo třídní metody, tak i statické metody. Tato třída obsahuje 2 veřejné atributy: char Color a int[4,2] Pozice. V tetrisu je možnost hrát se 7 různými tvary TetroBlocků, tedy se všemi kombinacemi, ke kterým dojdeme přehazováním 4 čtverečků (zanedbáme-li násobnost). Proto existuje 7 tříd, pro každý tvar zvlášť. Každá z těchto tříd je potomkem třídy Shape a musí obsahovat override metody pro pohyb doleva, doprava, dolů, oboustranné rotace a pohyb nahoru (pro účely ladění a nepovedený pokus o implementaci tetrisDFS). Protože logika rotací se u některých tvarů liší, existují pomocné soukromé funkce a atributy, které pomáhají chodu hry. Před každým pohybem nebo rotací se provede buď mateřská funkce pro kontrolu, zdali je možné se do tohoto místa pohnout, nebo kontrolní funkce příslušící danému tvaru. Pohyb nějakým směrem nebo kterákoliv rotace je možná v případě, že po dokončení akce bude možné TetroBlock zobrazit na hrací desce bez toho, aby byl překryt jakýkoliv čtvereček nebo aby figurka „vyletěla za hranice hrací desky“. Mateřská třída má ještě jednu podstatnou metodu a tou je: public int HardDrop(ref GameBoard gb). Vyvolání této metody je možné v uživatelském režimu Tetrisu zmáčknutím tlačítka SPACE. Jednoduše bude v cyklu probíhat funkce public bool MoveDown(ref GameBoard gb) dokud to bude možné a ve výsledku to vrátí počet úspěšných provedení této funkce. To číslo se pak přičte ke score uživatele. Je to praktické, v případě, že hráč už ví, kam chce figurku umístit a nechce několikanásobně mačkat tlačítko pro pohyb dolů nebo čekat. Zároveň získá body navíc, což je fajn bonus.

Třída Shape rovněž obsahuje statické metody, které jsou následně používány umělou inteligenci. Mezi tyto metody patří:

* static public bool Check[Left/Right/Down]Side(ref GameBoard gb, int[,] Pozice)
* public static bool MoveDownNotPossible(ref GameBoard gb, int[,] Pozice)

Druhá funkce je hojně používána při prohledávání stavového prostoru hrací desky, pro nalezení všech možných umístění TetroBlocku a zároveň i navigací k daným lokacím.

Kapitola 4 – form1.cs aneb třída, co vládne všem a všem káže

Form1 je nejdůležitější třída, protože se nacházíme ve formulářové aplikaci a všechny akce vychází z interakcí ve formuláři. Můžeme zde uvidět různé globální proměnné pro obecný chod hry nebo i specifičtější boolovské proměnné, které signalizují, který konkrétní režim právě běží a co se má anebo nemá vykreslovat na formulář. Mezi nejzákladnější obecné proměnné patří:

* Pen tuzka
* Shape activePiece
* Shape nextPiece
* int[] clearLines – označuje, které řádky jsou plně naplněny a můžou být vymazány
* bool gameover
* int moveSpeed

Jelikož bylo nemožné spolupracovat s událostmi jako Key\_Down Event nebo Key\_Press Event (nejspíš proto, že byl fokus vždy na nějakém tlačítku, a ne na PictureBoxu), musela se přepsat defaultní virtuální metoda, pro zpracovaní tlačítek ProcessCmdKey(ref Message msg, Keys keyData), která odpovídá za reakce při zmáčknutí tlačítka. V této funkci byl použit switch, který stisknuté akční tlačítko převedl na pohyb nebo jinou akci figurky.

Průběh života hrací figurky má několik fází. Nejprve se figurka vygeneruje a uloží do příslušné proměnné, na kterou vidí malovací události, které pak podstatné informace předávají funkcím na grafické znázornění. Hráč může pohybovat figurku všemi směry, nebo po uplynutí časové doby, v závislosti na obtížnosti, je figurka posunuta o jednu úroveň níž v události spojené s časovačem. V případě, že už není možné se posunout dolů, je figurka vložena správnou funkcí do hrací desky a následně přichází na řadu různé kontrolní funkce. Je zkontrolováno, zda nebyla naplněna nějaká řada a v pozitivním případě se řada vymaže, upraví se číselné hodnoty score, level, lines a moveSpeed, a nakonec se posune hrací deska a vygeneruje se nová hrací figurka.

## Část II – Umělá inteligence

Zápočtový program obsahuje umělou inteligenci, která je více či méně schopná hrát hru. Avšak není zde jenom jedna AI, ale dokonce 2. A nejsou si vůbec podobné! Jedna se jmenuje HardDropAI a jak už název napovídá, tak od této „inteligence“ nemůžeme čekat nějaké epické rotace a triky při hře. Vygenerovaná figurka se nejprve dostane do správné rotace a správné svislé polohy a následně jen klesá, dokud je to možné. Proto je název HardDropAI. Druhá má kratší název – AI, zato je mnohem efektivnější a „moudřejší“. Vygenerovaná figurka má možnost se dostat i POD nebo DO jiné figurky. Mimo jiné tato AI bere i v potaz následující figurku, tím pádem je vyzkoušeno a ohodnoceno mnohem více možností a z nich se vybere ta nejlepší. Při průběhu obou AI můžeme nalézt 3 různé fáze, které se cyklí stále dokola. Fáze:

1. Nalezení všech pozic, mezi kterými se budeme následně rozhodovat.
2. Bodový systém rozhodne, která pozice je pro dosavadní hrací desku optimální.
3. Máme už rozhodnuto a zbývá jenom „najít cestu od startovní pozice k cíli“.

Druhý krok je v obou AI téměř, až na nějaké drobnosti, identický. Ostatní kroky nemají skoro nic společného. Postupy se liší dramaticky, avšak i přes to používá druhá AI mnohé metody z třídy HardDropAI.

Druhá AI je časově náročnější, protože vyzkouší všechny kombinace aktuální i následující figurky. To znamená, že první dvě fáze umělé inteligence budou vykonány o řád vícekrát než v případě HardDropAI. Vyměníme čas za lepší rozhodnutí, což AI umožní déle hrát hru.

kapitola 1 – všechny cesty vedou na hrací desku

**HardDropAI**, jak bylo v předešlé části zmíněno, je sice „hloupější a slabší“, ale na druhou stranu je časově méně náročnýí. V hlavní třídě Form1.cs se nachází globální proměnná int[5,2] placeToDropFrom, která je výsledkem komplikovaného hledání nejlepší pozice, ze které po HardDropu získáme optimální výsledek. Optimální pozici dostaneme zavoláním statické metody ze statické třídy HardDropAI – placeToDropFrom = HardDropAI.FindBestPlaceForDrop(ref GameBoard gb, Shape shp);. Tato metoda zavolá ve svém průběhu 13 dalších metod. Všechny však nebudeme probírat do puntíku. Nyní nás zajímá jenom první řádek této funkce– int[,,] drops = findAllHardDrops(ref GameBoard gb, Shape shp); –, zbytek je popsán v dalších kapitolách. Tato metoda posune čerstvě vygenerovanou aktivní figurku nejvíc doleva a postupně zapisuje souřadnice TetroBlocku po HardDropu. Takhle se to opakuje, dokud nejsou vyzkoušeny všechny pozice ve všech rotacích. Hodnota drops[i, 0, 2] označuje kolikrát musí TetroBlock rotovat doprava, než se dostane do správného rozpoložení. Tato informace usnadňuje v budoucnu poslední fázi.

**AI** má tento krok mnohem složitější a propracovanější. V tomto případě používáme upravenou verzi BFS algoritmu pro prohledávání stavového prostoru. Stavový prostor je aktuální hrací deska. Při prohledávání využíváme frontu, která kromě informací o souřadnicích nese i navigace k danému místu od startu. Výsledkem prohledávání je uložení všech pozic do fronty (recyklujeme datovou strukturu), která kromě konečných pozic uchovává i „navigaci“ k dané pozici. To je velmi praktické a usnadní nám to poslední fázi. Prohledávání probíhá následovně:

1. Vybereme prvek z fronty (potřebujeme, aby metoda fronta.Pop() kromě souřadnic vrátila i navigaci, tedy klasický string, kde jednotlivé chary znamenají pohyb některým směrem nebo rotaci, proto je návratová hodnota objekt InfoBlock, což lze přirovnat k obyčejné struktuře o 2 proměnných – pole se souřadnicemi a string s navigací).
2. Pokud už pohyb dolů není možný (jsme na kraji desky nebo se pod TetroBlockem již něco nachází), tak pozici (včetně navigace) uložíme do výsledné fronty.
3. Zkusíme naše souřadnice posunout všemi směry a úspěšně posunutí vložíme do fronty s modifikovanou navigací. Vložené místo „označkujeme“, aby už nebylo možné do této pozice vstoupit, čímž zabráníme zacyklení prohledávání nebo zbytečně vysoké časové obtížnosti.

Tento tetris-BFS algoritmus provedeme ve všech rotacích. Nyní máme frontu naplněnou všemi pozicemi ve všech rotacích, ke kterým můžeme dojít. Můžeme se pustit do vyhodnocení nalezených pozic.

V programu můžeme nalézt pokus o tetris-DFS algoritmus, který však se nevyvedl, a proto není v programu používán.

kapitola 2 – dobrý tah, nebo zlý, to je, oč tu běží

Druhá fáze obou inteligencí je, až na jedno kritérium, nerozeznatelná. Každá nalezená pozice se hodnotí v 6 disciplínách: počet zablokovaných bloků „natvrdo“, počet zablokovaných bloků „nalehko“, rozdíl výšek hrací desky a nejvyššího bloku figurky, počet naplněných řádků a rozdíl „hrbolatosti desky“, rozdíl počtu hlubokých sloupců. Pro každé kritérium existuje vlastní metoda, která navrátí číselnou hodnotu odpovídající danému kritériu. Začátek všech metod je stejný. Nejprve dosadíme souřadnice kandidáta na nejlepší pozici a vynulujeme řádky, které by byly naplněny. Pokud je třeba, tak posuneme hrací desku. Následně provedeme výpočet hodnot, konstantami vynásobíme výsledné hodnoty a v případě, že výsledek má nižší číselnou hodnotu než dosavadní nejlepší pozice, tak danou pozici uložíme jako zatím nejlepší.

**Počet zablokovaných „natvrdo“**. Tento pojem je definován jako čtverec v hrací desce, ke kterému nelze dojít ze startu klasickým prohledáváním do šířky. Proto je zde implementován obyčejný BFS, kde vrcholy jsou všechny souřadnice a hrany mezi vrcholy existuje, v případě, že oba vrcholy jsou neobsazené (‘\0‘). BFS je spuštěno pod všechny čtverečky vložené kandidátní figurky, které nejsou vymazány a je pod nimi neobsazené místo. V případě naplněných řádků je spočítáno posunutí souřadnic. Jestliže BFS dojde na nejvyšší řádek desky, je BFS přerušeno a není nalezeno žádné „zablokování“. Metoda vrátí počet zablokovaných bloků v herní desce.

Toto je jediná metoda, která je v 2. AI pozměněná. Průběh výpočtu je stejný, avšak jsou kontrolovány nejenom souřadnice POD figurkou, ale i vedle figurky. To proto, že 2. AI se umí pohybovat i do stran a figurka může být dosazena na místa, ke kterým HardDropAI nedojde.

Tato informace patří k jedné z nejdůležitějších, protože při častém blokování dochází k nežádoucímu růstu hrací desky a zvyšování obtížnostní pro naplnění řádku, čímž se pomalu, ale jistě přibližujeme k prohře.

**Počet zablokovaných „nalehko“**. K tomuto výpočtu dojde jen v případě, že první kritérium má nulovou hodnotu. Kontroluje se jenom počet neobsazených bloků pod dosavadní figurkou. Nemusí se tedy vůbec jednat o nepřístupnou pozici, avšak i přesto je tento jev méně žádoucí. Je-li tedy možnost zahrát jinak a zachovat ostatní parametry na stejných hladinách, tak změníme cílovou pozici.

**Rozdíl výšek hrací desky a dosazené figurky**. Není žádným tajemství, že lepší tah je ten, který nezvyšuje výšku hrací desky. Čím níž dosadíme figurku, tím víc se nám zvyšuje šance na naplnění nějakého řádků a zachování „zdravé podoby desky“. Hned po prvním kritériu je toto nejdůležitější, protože čím níž se budeme snažit pohybovat, tím víc se budou naplňovat řádky.

**Počet naplněných řádků**. Smysl celé hry je naplnit co nejvíce řádků a tím pádem zvýšit úroveň a score. Avšak při rozhodování o cílové pozici, paradoxně, nemůže být toto nejdůležitější faktor. Kdyby tomu tak bylo, AI by hrála jak splašený kůň a vybírala by cílovou pozici jenom protože zde našla možnost pro naplnění řádku. Tato strategie není vůbec efektivní a přínosná.

**Rozdíl hrbolatosti**. Přestože je název velice zvláštní, jedná se o velmi důležitou hodnotu. Nejprve jsou vypočítány výšky všech sloupců a následně je vypočítána hrbolatost herní desky, což je součet absolutních hodnot rozdílů výšek sousedních sloupců. Pak se přidá herní figurka a je znova spočítána hrbolatost. Metoda vrátí rozdíl těchto hodnot. Pří tomto výpočtu je použitá knihovní funkce Math.Abs. Vysoká hrbolatost desky překáží efektivnímu naplňování řádků.

**Rozdíl hlubokých sloupců.** Výpočet této hodnoty je velmi podobný předchozímu faktoru, avšak hodnotí se něco lehce jiného a konstanta u závěrečného výpočtu se dramaticky liší. Spočítá se počet „hlubokých sloupců“ před a po dosazení figurky a rozdíl těchto hodnot pak určí, zdali je pozice vhodná či nevhodná. Hluboký sloupec lze definovat jako sloup v hrací desce, který je obklopen sloupy jenž jsou vyšší alespoň o 3 bloky. Tyto tvary jsou velmi nepříznivé a ničí „zdravému tvaru hrací desky“.

int tempScore = hardBlocked \* 9 + softBlocked \* 7 + diff \* 5 – numLines \* 3 + hrbolatost / 4 + emptyPillars \* 10;

kapitola 3 – když jde tetroblock na výlet

Nyní se konečně dostáváme k poslední fázi. Máme figurku, máme cílovou pozici a pomocné informace. Obě AI používají různé způsoby pro dosažení cíle.

**HardDropAI** má krapet složitější „navigaci“ k cíli. V hlavní třídě Form1 je uložena pozice, ze které bude následně figurka dolů jenom „padat“. V této pozici je taky uložen počet potřebných rotací. Poté je nalezen směr, kterým se figurka musí vydat a po dosažení souřadnic začne figurka „padat dolů“. Globální proměnná int[5,2] placeToDropFrom obsahuje prvek [4,0] , který může mít následující hodnoty:

* >0 && <4 – označuje počet rotací doprava
* 0 – označuje, že figurka dorotovala a je třeba najít směr, kterým se hned vzápětí vydá
* -100 – pohyb doleva
* 100 – pohyb doprava
* 200 – pohyb dolů

V případě, že metoda static public bool HardDropAI.PlayNextMove(ref GameBoard gb, Shape shp, int[,] finishPoz) vrátí false, dostane AI signál, že je již figurka v cíli a může začít opět v první fázi s novou figurkou.

**AI** má tento krok mnohem jednodušeji. Jak už bylo zmíněno dříve, při prohledávání a vyhodnocování pozic, je vždy k dostání navigace v stringové proměnné, kde jednotlivé chary označují pohyb nějakým směrem, anebo rotaci. Pří nalezení nejlepší pozice je tato navigace uložena v hlavní třídě Form1. To znamená, že si program musí zaznamenat aktuální krok navigace, což zvládne obyčejná proměnná typu int. Po dokončení navigace, začne AI opět prohledávat stavový prostor hrací desky a snažit se o získání nejlepší pozice. Například „GGDDRD“ označuje 2 rotace doprava, 2 pohyby dolů, pohyb doprava, a nakonec pohyb dolů.

# Uživatelská sekce

Nacházíme se ve formulářové aplikaci, tudíž nejlepší způsob jak ovládat nynější mód je skrz tlačítka. Nalezneme jich celkem 5.

1. Wall Breaker – zapne hru Wall Breaker a resetuje statistické údaje
2. Tetris – zapne hru tetris a resetuje statistické údaje
3. Exit – vypne hru
4. Hard Drop AI – zapne „slabší“ umělou inteligenci a resetuje statistické údaje
5. Improved AI – zapne „silnější“ umělou inteligenci a resetuje statistické údaje

**Tetris**, jak bývá zvykem, má jednoduché ovládání. Pohybujeme s aktivní figurkou po hrací desce a v případě nemožnosti se posunout dolů se na začátku hrací desky objeví figurka, která doposud byla znázorněna jako následující figurka. Hned vzápětí se vygeneruje jiná následující figurka, která bude, stejně jako předchozí, zobrazena na stejném místě. Hru tetris ovládáme následujícím způsobem:

* Šipka doleva – posune figurku doleva, je-li to možné
* Šipka doprava – posune figurku doprava, je-li to možné
* Šipka dolů – posune figurku dolů, je-li to možné, v opačném případě „spawne“ novou figurku na startu hrací desky
* Šipka nahoru – provede rotaci ve směru hodinových ručiček, je-li to možné
* Z – provede rotaci proti směru hodinových ručiček, je-li to možné
* SpaceBar – provede HardDrop figurky, což je pohyb dolů, dokud nenarazíme na kraj nebo jiné bloky
* Existuje i tlačítko pro pohyb nahoru, ale to je před uživatelem skryto :)

**Bodový systém tetrisu** je následující. V případě, že hráč provede HardDrop, získá jeden bod za každý pohyb dolů. Pří naplnění řádků jsou body určeny počtem řádků a nynějším levelem. Za 1 řádek hráč získá 40 bodů, za 2 získá 100, za 3 získá 300 a za 4 získá 1200. Toto číslo je k tomu vynásobeno aktuálním levelem, tudíž čím déle hráč hraje, tím větší získává body za naplněné řádky. Po každém naplnění 10 řádků se zvýší úroveň, čímž se zvyšuje tempo hry. Na 10. úrovni je tempo 140ms/tah, což je strop a níž už rychlost hry nepůjde.

**Wall Breaker** má ovládání mnohem jednodušší. Pohyb doleva a doprava je určen stejně jak v tetrisu a SpaceBar umožní hráči střílet po padajících kostkách. Pokud padající kostky dojdou až k samotnému hráči, hra končí.

**Bodový systém WB** je ještě jednodušší. Za zničenou kostku získá hráč jeden bod. Každých 20 bodů se zvýší úroveň a zvýší rychlost padání kostek o 30 ms.

# Závěr

Daný zápočtový program lze rozdělit na 2 hlavní části. První část je samotná hra Tetris, její ovládání, zobrazování, logika hry a bodovací systém. Druhá část se soustředí na umělou inteligenci a způsob jak donutit počítač (vhodnými algoritmy) hrát efektivně tetris a získat co nejvíce bodů. Při umělé inteligenci nalezneme různými algoritmy všechny možné výsledné pozice a statickými ohodnocovacími funkcemi AI zjistí, která poloha je nejvhodnější. Hodnocení probíhá v 6 kritériích a následně se vynásobí konstantami. Tyto konstanty byly získány experimentální cestou, avšak je jisté, že existuje mnohem lepší varianta. Úloha nalezení optimální konstant je ponechána čtenáři.