**PIŠKVORKY – GOMOKU**

### Dokumentace k zápočtovému programu

#### Kryštof Žucha

# Obsah

Obsah 2

Úvodní informace o práci 3

Zadání 3

Konkrétní splněné cíle 3

Uživatelské informace 4

O hře 4

Ovládání programu 5

Technické informace 6

Technologie 6

Součásti aplikace 8

Kód hlavního okna 9

# Úvodní informace o práci

Zde naleznete zcela obecné informace o mé práci. Je zde vypsáno konkrétní zadání a finální realizace, a to včetně úplného seznamu použitých technologií.

## Zadání

Cílem práce je vytvořit hru piškvorky v jazyce Pascal a formulářového frameworku. Program má obsahovat umělou inteligenci, tedy počítačového protivníka, proti kterému bude možné hrát.

## Konkrétní splněné cíle

K tvorbě jsem dle zadání použil Pascal. Psal jsem v IDE Lazarus, které obsahuje i formulářový framework (Lazarus forms) postavený na Win32. Jedná se o aplikaci, která se velmi podobá Borland Delphi, ale je stále aktuální (stále se vyvíjí) a navíc s ní pracovali přednášející i cvičící, proto jsem si ji zvolil.

Jednoduchou ikonku jsem vytvořil v Microsoft Paint, neboli v Malování. Jednoduchý rastrový editor na tuto práci bohatě stačí. Výstupní obrázek jsem převedl na formát .ico pomocí online služby www.convertico.com.

Aplikace nakonec sestává z hlavního okna, v jehož horní části je jakési „hlavní menu“, kde se dá hra nastavit. Ve spodní části je pak samotná herní mřížka a pomocné kontrolky pro ovládání průběhu hry. Více o nich zmíním v části dokumentace o ovládání aplikace.

Další pomocné okno obsahuje doplňková nastavení hry, vyvolává se jako dialog z hlavního okna. Navíc také některé akce vyvolávají standardní dialogová okna s informací či dotazem.

# Uživatelské informace

## O hře

Piškvorky, originálně gomoku, je konečná hra dvou hráčů původem z Japonska. Jedná se o oblíbenou zábavu pro všechny generace, v posledních letech se navíc mj. pořádají středoškolské turnaje po celé České republice. Anglicky se o ní mluví také jako o gomoku, slovním spojením tic‑tac‑toe se rozumí zjednodušená hra na mřížce 3x3. Existuje také varianta ultimate tic‑tac‑toe, která se hraje na poli 3x3 mřížek 3x3 a má specifická pravidla.

Tradiční gomoku se hraje na čtverečkové mřížce 19x19. Hráči hrají s bílými a černými kameny, které pokládají na místa křížení jednotlivých čar.

Piškvorky na papíře se hrají též ve čtverečkové mřížce. Hráči mají symboly křížku a kolečka, obvykle modré a červené barvy, které vepisují do buněk, které mřížka tvoří.

Cílem hráčů, ať už mají kameny či píší symboly, je vytvořit v herním poli tzv. piškvorku, čímž se rozumí nepřerušená řada 5 svých sousedících symbolů, a to v řádku, ve sloupci či na některém diagonálním směru. Klasické gomoku obvykle nedovoluje vytvořit delší řadu, tedy pokud by k tomu došlo, tak hráč, který takovou řadu vytvořil, prohrává.

Hráči tedy musí zároveň tvořit sousedící řady svých symbolů a zároveň v tom bránit soupeři. Hráč, který danou partii zahajoval, má v tomto směru jistou výhodu. Této výhodě se někdy brání speciálními pravidly, kdy hráč, který začíná, umístí 2 a 1 herní symbol předem do herního pole, a následně si druhý hráč vybere, se kterým symbolem bude danou partii hrát.

Samozřejmě je možné hru dále upravovat, např. časovým limitem, za který musí hráč provést svůj tah, celkovým časovým limitem partie, velikostí herní mřížky apod. Vše je v rukou hráčů, kteří by se na případných pravidlech měli dohodnout před zahájením hry.

## Ovládání programu

Program je členěn na dvě části. Nad horní částí se navíc nachází nápis „Piškvorky“ a vedle něj tlačítko **i**, které zobrazí podobné informace o programu, jako ty napsané v tomto dokumentu. Poskytují stručné informace o hře a zjednodušený popis ovládání programu.

Když je program spuštěn, je vždy aktivní právě jedna z částí programu. Aktivní část je podkreslena zeleným pozadím. S neaktivní částí nelze pracovat, je „zašedlá“ a ovládací prvky nereagují.

Horní část programu tvoří hlavní menu. Zde se nastavují veškeré parametry hry před jejím zahájením. Každému hráči je možné dát jméno, to slouží pro zobrazování informací v průběhu hry. Jméno samozřejmě není povinné, takže pokud není zadáno, zobrazí se náhradní text „Hráč 1“ či „Hráč 2“. Dále se nastavuje typ hráče, buďto skutečný, lidský hráč, nebo počítačem řízený hráč. Také se musí zakliknout, který z hráčů začíná první kolo.

Tlačítko „Možnosti“ zobrazuje další nastavení hry. Je možné zvolit chytrost umělé inteligenci počítačem řízených hráčů, dále limit délky tahu hráčů a způsob výběru hráče, který začíná v dalším kole. Navíc se nahoře nachází dvě tlačítka pro práci s uloženým nastavením.

Program vždy při ukončení ukládá nastavení hry do konfiguračního souboru. Pokud si přejete obnovit výchozí nastavení programu, stiskněte tlačítko „RESET“. Pokud si přejete funkci ukládání vypnout, nastavte si program do vámi preferovaného výchozího stavu a stiskněte tlačítko „Uložit výchozí stav“. Pro změnu vlastního výchozího nastavení jej použijte opakovaně. Pokud si přejete funkci ukládání opět povolit, je třeba opět stisknout tlačítko „RESET“. Obě tlačítka navíc po použití vyžadují restartování programu.

Dolní část programu tvoří samotnou hernu. V levé části se nachází mřížka 15x15, na kterou se umisťují symboly. To se provádí stisknutím levého tlačítka myši na příslušné buňce. Poslední symbol umístěný do herního pole je vždy podbarven žlutou barvou, pro přehlednost.

Napravo od mřížky se nachází informační panel, na kterém se zobrazuje čas kola, skóre hráčů, kdo je další na tahu a čas tahu hráče. Ten je důležitý, pokud je nastaven časový limit tahu, viz. odstavec o tlačítku „Možnosti“ výše. Pokud je limit nastaven, zobrazuje se.

Pod informačním panelem také naleznete tlačítka sloužící k ovládání průběhu hraní. Tlačítko „Vzdát se“ ukončí předčasně kolo a body získává hráč, který tlačítko nezmáčknul. Program se po stisknutí ptá na potvrzení.

Tlačítko „Nabídnout remízu“ se nejprve zeptá hráče, který tlačítko nezmáčknul, jestli přijímá remízu, a v případě jeho kladné odpovědi kolo předčasně ukončí bez bodového zisku.

Tlačítko „Nové kolo“ zahájí nové kolo. Naopak tlačítko „Skončit“ ukončí hru a program se vrátí do hlavního menu.

# Technické informace

Zde naleznete popis programátorské stránky aplikace, popis použitých technologií a algoritmu umělé inteligence.

## Technologie

Konkrétní použité technlogie již byly jmenovány, rozeberu zde Lazarus forms. Podstatné je, že se jedná o formulářový framework postavený na Win32, který stojí na Objektově orientovaném programování. Každý pascalový zdrojový soubor tak musí v hlavičce obsahovat direktivu {$mode objfpc}, kterou se freepascal „přepne do objektového režimu“. Formulářem se rozumí okno aplikace, jaké známe z většiny běžných aplikací pro systém Windows.

Každé okno aplikace sestává ze dvou souborů – soubor designeru a zdrojový kód formuláře, tzv. codebehind. Soubor designeru má příponu .lfm (Lazarus ForM) a je generován přímo Lazarusem, na základě toho, jak je formulář nadesignován v interaktivním okně designeru v Lazarusu. Toto okno představuje jakýsi „náhled“, jak bude konečný vykreslený formulář vypadat. V okně designeru je možné jednotlivé kontrolky přesouvat, mazat a přidávat je. Jejich vlastnosti (jako je velikost, barva pozadí...) lze po nakliknutí (označení) v okně designeru měnit v okně Inspektor Objektů.

Soubor zdrojového kódu má standardní příponu .pas a obsahuje kód pro manipulaci s formulářem. V jeho hlavičce se nachází definice tzv. třídy objektu formuláře, což je jakýsi „výpis“ všech procedur, funkcí, proměnných (a vlastností), které formulář obsahuje. Část proměnných jsou samotné kontrolky formuláře, jejich definici jako proměnných zde generuje přímo Lazarus. Naopak jejich vlastnosti (jednotlivých kontrolek) se nachází právě v souboru designeru, a jejich nastavení se tak překládá z tohoto souboru.

Ještě popíšu, co jsou to vlastnosti (neboli properties). Navenek (při práci s objektem jinde ve zdrojovém kódu) se tváří jako proměnné, ovšem interně obsahují proceduru setter (deklaruje se klíčovým slovem write) a funkci getter (deklaruje se klíčovým slovem read). Setter dostane v prvním parametru novou hodnotu (podle datového typu vlastnosti, stejné jako u proměnné), kterou se program snaží do vlastnosti uložit, a má za úkol tuto hodnotu uložit. Obvyklé je, že objekt obsahuje interně nějakou proměnnou, do které se hodnota ukládá, a setter ji uloží. Setter ale může například ověřit, že se program nepokouší uložit nepovolenou hodnotu a přizpůsobit se tomu. Dále, pokud uložení nové hodnoty má např. vyvolat změnu v jiném místě aplikace, může ji setter snadno vyvolat. Druhá část vlastnosti je getter, je bezparametrický a má za úkol vrátit hodnotu vlastnosti. Tedy v nejjednodušší formě vrátí obsah interní proměnné.

Pascal má tu výhodu, že pokud se vlastnost opravdu jen odkazuje na nějakou třídní proměnnou, ale neobsahuje žádnou přidruženou logiku, je možné v deklaraci předat pouze název pomocné proměnné, bez nutnosti definovat příslušnou funkci či proceduru.

Důležitým prvkem Objektově orientovaného programování jsou také tzv. události. Jedná se o určitou vlastnost, kterou může objekt mít. Tato vlastnost obsahuje pointer na tzv. obslužnou proceduru, což je procedura, která se volá, jakmile událost nastane. To, že nějaká událost nastane, objekt pozná sám, tím, že je s ním manipulováno nebo naopak sám s něčím manipuluje. Nějaká manipulace je právě vázána na událost. Pokud události přiřadíme nějakou obslužnou proceduru, je volána vždy, když k manipulaci dojde, a v obslužné proceduře tak může aplikace na danou událost reagovat. Konkrétním typickým případem je např. obsluha události Click na tlačítku, která se volá vždy, když uživatel na tlačítko klikne.

Program používá tzv. dynamické vykreslování. Řada kontrolek obsahuje tzv. „Canvas“, česky plátno, na které je možno něco nakreslit. Kreslení se provádí zavoláním příslušné procedury na Canvasu, např. Canvas.Line(), Canvas.Ellipse() apod. těmto procedurám se předávají parametry o pozici a barvě kýženého útvaru, který se má vykreslit. Kód, který má vykreslit dané prvky, se umisťuje do obslužné procedury události OnPaint, která se volá na kontrolce s Canvasem pokaždé, když se musí překleslovat znovu. To se děje, když je pohnuto s oknem aplikace, okno je zvětšeno či zmenšeno či maximalizováno z lišty. Pokud chceme překleslení vyvolat sami, zavoláme na kontrolce s Canvasem proceduru Invalidate. To se hodí, konkrétně v této aplikaci se tak děje pokaždé, když je umístěn nový symbol do herní mřížky, když uživatel pohybuje myší v herní mřížce, přechází-li se z menu do hry a naopak nebo když se změní hráč, který je na tahu.

## Součásti aplikace

Zdrojový kód aplikace má několik souborů. Hlavním souborem je piskvorky.lpr, což je výchozí soubor s programem. Nachází se zde vstupní bod aplikace, který vytvoří formuláře a aplikaci spustí. Tuto část kódu generuje přímo Lazarus. Dále se zde nachází procedury pro čtení a zápis konfiguračního soboru, které se volají při spouštění a ukončování aplikace.

Za zmínku stojí funkce (s tímto konkrétním parametrem) GetEnvironmentVariable('appdata'), která vrací systémovou cestu do složky s konfiguračními soubory aplikací. Tím je zajištěno, že se ukládání a načítání konfigu děje v systému ze složky, která je pro toto určená, takže uživateli nebude soubor překážet v dokumentech, neznalý uživatel jej ani nenajde a omylem nezmodifikuje a nebudou nastávat výjimky související s tím, že by se program pokoušel pracovat se souborem ve složce, ke které nemá přístupová práva.

Další soubor je Global.pas, zde se nacházejí globální konstanty, které program potřebuje. Konkrétně obsahuje text okna, které se zobrazuje tlačítkem **i** a výchozí hodnoty konfiguračního souboru. Dále obsahuje globální proměnné, pomocí kterých se načtené hodnoty konfiguračního souboru dostanou k hlavnímu formuláři. Po načtení hodnot ve vstupním bodu aplikace je program uloží do těchto proměnných, odkud si je pak formulář ve správný čas vezme. Pro ukládání se tyto nepoužívají, program si hodnoty bere přímo z kontrolek. Když se ale aplikace spouští, formulář ještě není v paměti vytvořen, a neexistuje tedy způsob, jak kontrolky nastavit přímo po načtení konfigu.

V souboru Default\_AI.pas je kód umělé inteligence, tedy počítačového protihráče. Obsahuje funkci GetName, která vrací „jméno“ umělé inteligence, které se zobrazuje v menu při výběru typů hráčů. Dále proceduru Move, která má dva out parametry, do kterých procedura uloží souřadnice tahu. O použitém algoritmu více viz Algoritmus Umělé Inteligence.

Aplikace obsahuje dva formuláře, a to gomoku\_board a options. První zmíněný je hlavní okno aplikace, a jeho zdrojový kód je defacto zdrojovým kódem celé aplikace, takže mu věnuji samostatnou kapitolu, viz Kód hlavního okna. Druhý zmíněný je okénko vyvolané stiskem tlačítka Možnosti, a obsahuje čtveřici kontrolek s dalšími nastaveními. S jejich hodnotou se pracuje přímo z hlavního okna aplikace, a to na základě stisku tlačítek, které okénko zavřou. Jedná se o tlačítka OK a Storno. Tato mají přenastavenou vlastnost ModalResult, což v podstatě značí, že jejich stisk zavře okno. Více viz Kód hlavního okna. Tlačítka pro Reset či Uložení výchozích hodnot provedou zápis do globálních proměnných a pak hned vyvolají ukončení aplikace, bez návratu na hlavní formulář. Při ukončování pak aplikace na základě globálních proměnných pozná, že uživatel tlačítko stisknul.

## Kód hlavního okna

Jelikož se v kódu okna nachází naprostá většina funkčního kódu aplikace, je soubor rozčleněn na několik sklapnutelných regionů (deklarují se direktivou {$Region JMENO\_REGIONU}, ukončují direktivou {$EndRegion}) . Tyto regiony pak sdružují proměnné, procedury a funkce, které spolu logicky souvisí. Nad každou položkou, která nespadá pod obsluhu nějaké události, je v komentáři stručný popis, co to je a k čemu to slouží (v anglickém jazyce).

V horní části souboru se nachází deklarace třídy formuláře, a vygeneroval ji z naprosté většiny přímo Lazarus. Ručně jsou přidány pouze čtyři třídní procedury, které slouží k obsluze samotného procesu hry, jejich definice jsou v regionu Game Flow Procedures.

V regionu Common Game Variables se nachází třídní proměnné, které není potřeba reprezentovat jako vlastnosti. Nejdůležitější z nich jsou GameField, což je pole polí bytů, a představuje samotné herní území. Samotné pole polí bytů je jako nový datový typ GameFieldArray deklarováno v souboru global.  
0 – neobsazené pole  
1 – pole obsazené hráčem 1 (kolečka)  
2 – pole obsazené hráčem 2 (Křížky)  
  
Dále region obsahuje řadu proměnných uchovávajících souřadnice v herním poli, jedná se o souřadnice umístěných symbolů, souřadnice pozice kurzoru či souřadnice, mezi kterými je výherní kombinace. Dále proměnné, které uchovávají čas spuštění časovačů (viz region Timer Handlers) a proměnná IsP1ToStart, což je boolean, který určuje, jestli další hru zahajuje hráč 1 či hráč 2.

V regionu property store vars se nacházejí proměnné, které používají vlastnosti pro ukládání svých hodnot. Vyznačují se tím, že na začátku svého názvu mají pro snadné odlišení znak podtržítka.

Další dva regiony obsahují deklarace vlastností a jejich setterů. Region Game State Properties obsahuje vlastnosti, které zachycují stav herního pole a průběhu hry. Skóre hráčů, aktuální čas časovačů, kdo je na tahu a index čtverečku herního pole, nad kterým je kurzor. Všechny tyto hodnoty se nějakým způsobem vykreslují na obrazovku, proto jsou uchovávané jako vlastnosti.

Region Game Settings Properties obsahuje vlastnosti, které představují nastavení hry, která se ukládají do konfiguračního souboru, a jsou to tedy povahově typická data, která je vhodné reprezentovat vlastností. Mají dopad na vykreslení, a herní mechanismy si z nich potřebují číst, jak se mají chovat.

Posledním regionem hlavičky je Constants, a obsahuje konstanty, které jsou relevantní pro okno aplikace. Jedná se pouze o barvy částí aplikace, které se v kódu používají opakovaně (jako barva konkrétního hráčského symbolu) a je tedy vhodné deklarovat si je na jednom místě pro případné změny.

Dále následují regiony v těle třídy okna formuláře, a obsahují už jen funkce a procedury. Kromě zmíněného Game Flow Procedures se vždy jedná o obsluhy událostí.

První z nich jsou Game State Property Setters a Game Settings Property Setters, a obsahují definice setterů pro konkrétní vlastnosti. Setter vždy nejprve podmínkou ověří korektnost zapisované hodnoty a pak ji zapíše. Některé settery navíc volají proceduru Invalidate, neboť jejich změna implikuje překleslení dynamických Canvasů.

Region Window Handlers obsluhuje přímé události okna aplikace. Událost OnCreate, která nastane, když je okno zkonstruováno a připraveno k zobrazení, a dělají se zde přípravné práce. Zinicializuje se array herního pole a do vlastností nastavení se dosadí z globálních proměnných a do všech ostatních se dosadí výchozí hodnoty. Podle hodnot vlastností se pak korektně nastaví všechny kontrolky na formuláři do kýženého počátečního stavu.

Procedura DefocusControls přenastaví zaměření klávesnice na prázdný panel, který se na formuláři nachází. Volá se při kliknutí na hlavní okno a na panely v něm (tedy z uživatelského pohledu do prostoru mimo kontrolky), a uživatel se tím „vyklikne“ z momentálně zaměřené kontrolky.

Region OnPaint Handlers obsahuje dynamické vykreslování. Obsahuje proceduru pro každou komponentu, do jejíhož Canvasu se něco kreslí. Vykreslení se provede podle souřadnic, které jsou pevně dané tvarem okna, souřadnic, které se vypočítají (u herního pole) a barvami, které jsou definované v regionu Constants.

Regiony Menu Controls Click Handlers a Game Controls Click Handlers obsluhují kliknutí na jednotlivé kontrolky v okně. Menu se týká horní části programu, Game dolní. Naprostá většina kontrolek horní části nastavuje příslušné vlastnosti. Tlačítko **i** zobrazí dialog s textem z globální proměnné, tlačítko Start volá proceduru StartNewGame a tlačítko Možnosti zobrazí rozšířená nastavení. Provede to pomocí funkce ShowModal, která vrací celé číslo. Toto číslo reprezentuje ModalResult zavolaného dialogu, který se nastaví podle „Modal“ významu tlačítka, jež dialog ukončilo, více viz Options.pas. Modal význam tlačítka OK je 6, takže pokud dialog skončil tlačítkem OK, ShowModal vrátí 6 a podle toho jsou buďto hodnoty z kontrolek uloženy do vlastností nebo naopak kontrolky zpětně znovu podle vlastností nastaveny.

Kontrolky v dolní části programu jsou úzce provázány s průběhem hry, takže společně s procedurami pro obsluhu hry (Game Flow Procedures) ovládají herní události v okně.

Region GameFieldPanel Mouse Handlers obsluhuje veškeré události týkající se myši na Panelu, do kterého se dynamicky vykresluje herní pole. MouseMove a MouseLeave nastavují pozici myši nad herním polem podle indexu buňky do příslušné vlastnosti. Click ověří, zdali je lidský hráč na tahu a jestli už nekliknul v tomto kole podruhé (pro případ, že by to hráč stihnul před dokončením obsluhy), a pokud je vše v pořádku, nastaví v proměnné index buňky, kterou hráč zvolil, podle pozice kurzoru. Více o průběhu hry viz popis Game Flow Procedures.

Region Timer Handlers obsluhuje událost Timer obou časovačů (Timerů). Tuto událost vyvolává každý běžící Timer v příslušném intervalu. Obě obsluhy jednak aktualizují příslušnou vlastnost s aktuálním časem a druhak průběžně pomáhají s obsluhou průběhu hry. Individual Timer ověřuje, zdali lidský hráč nepřekročil časový limit tahu, Main Timer ověřuje, zdali lidský hráč táhnul.

Poslední region Game Flow Procedures jsou procedury obsluhující průběh hry. Zde je také detailní popis řízení průběhu hry. Každá hra (každé kolo) začíná voláním StartNewGame, která zresetuje herní pole a příslušné vlastnosti a spustí časovače. Pokud se procedura volá z hlavního menu, je předtím nastavena boolean vlastnost GameState na true, aby se program z menu „přesunul“ do herního území. To provádí setter této vlastnosti - zpřístupní a zakáží se příslušné kontrolky. Podle GameState se samozřejmě řídí i dynamické vykreslování.

Jakmile je herna připravena, zjišťuje procedura StartNewGame, jestli je na tahu umělá inteligence, pokud ano, provede volání prvního tahu na danou umělou inteligenci (jak přesně tah AI probíhá viz níže). Jinak se hra dostane do stavu, kde se čeká na tah lidského hráče.

Pokud se čeká na tah od hráče, ať už lidského nebo počítače, znamená to, že MainTimer sleduje, zdali byly zapsány nějaké souřadnice tahu do příslušné proměnné. Výchozí hodnotou jsou souřadnice mimo index herního pole, jakmile se změní, je jasné, že bylo táhnuto. U lidského hráče je zapisuje obsluha kliknutí na panel s herním polem, viz popis regionu GameFieldPanel Mouse Handlers.

Tah Umělé Inteligence funguje tak, že se volá procedura Move na umělé inteligenci. Té se jako out parametr předává právě proměnná souřadnic aktuálního tahu. Jakmile se tedy zavolá, vyvolávající procedura už nemá s tahem umělé inteligence nic společného, vykoná se jakoby nezávisle. Jakmile umělá inteligence zapíše souřadnice svého tahu, nezávisle na ní si toho všimne MainTimer a tah je zpracován, stejně jako u lidského hráče.

Jakmile MainTimer zjistí tah, nastaví si proměnnou GamePlayerTurnProcessed na true, aby v příštích obsluhách svého eventu nezpracovával stejný tah vícekrát, a následně zavolá proceduru ProcessMove, která provádí samotné zpracování tahu. Podívá se, jestli byl tah validní (na prázdné existující pole) a umístí symbol táhnoucího hráče do daného pole. Dále provede detekci výhry, pokud někdo vyhrál, volá se procedura EndGame, jinak se pokračuje detekcí, zdali již není celé pole plné symbolů. Pokud ano a nikdo tedy nevyhrál, volá se procedura TerminateGame. Jinak se přehodí hráč na tahu, vyresetují se proměnné hráčova tahu a IndividualTimeru, a pokud je hráč počítačový, volá se příslušná procedura Move.

IndividualTimer, jak bylo řečeno, sleduje překročení času na tah. Pokud je čas překročen, volá proceduru TerminateGame. Tato procedura bere jako parametr číslo hráče, který má vyhrát, pro případ remízy pak číslo 0 (pro případ volání po zaplnění pole, viz výše). Podle tohoto parametru nastaví proměnnou aktuálního hráče na tahu a zavolá proceduru EndGame.

Procedura EndGame vypne časovače a nastaví, kdo bude začínat následující kolo, podle vlastnosti s příslušným nastavením a proměnné, ve které je stále zapsáno, kdo začínal právě ukončované kolo. Také připočte výhru hráči, který je aktuálně na tahu.

## Algoritmus Umělé Inteligence

Umělá inteligence používá heuristický algoritmus, který si podle aktuálního stavu herního pole ohodnotí volná políčka v herním poli a na základě hodnot vybere, kam táhnout. Algoritmus podle mých zkušeností hraje dobře, i když vyhodnocuje pouze aktuální stav (nedívá se, jak se hra bude vyvíjet několik tahů předem), takže jsem se nepustil do MiniMaxu, který je složitější a i náročnější na výpočet.

Vyhodnocení políčka probíhá postupně pro každý ze čtyř směrů (shora dolů, zprava doleva, zprava nahoře doleva dolů a zleva nahoře doprava dolů).

Pro každý směr jdeme vždy dvakrát (tj. pro první směr nejprve nahoru, pak dolů), vždy začínáme na vyhodnocovaném políčku a postupujeme směrem z něj. Pokud narazíme na náš znak, připočítáme si jedna do proměnné count a jdeme dál. Pokud narazíme na soupeřův znak, hledání v daném směru ukončíme a připočteme si jedna k proměnné barrier. Pokud narazíme na prázdné políčko či hranici indexů herního pole, hledání v daném směru ukončíme. Hledáme maximálně do vzdálenosti 5 (na každou stranu) od vyhodnocovaného políčka.  
Nyní máme v proměnné count počet našich políček napřímo sousedících s políčkem a v proměnné barrier 0, 1 nebo 2 podle izolovanosti políčka. Pokud count přesahuje 3, změníme jej na 20 (to je kvůli výhernímu políčku, aby obloženější políčko s kratšími řadami smybolů nedostalo přednost). Dále count umocníme na 2 a provedeme připočtení k hodnotě políčka. Pokud je obtížnost nastavena na nejlehčí, připočteme přímo count. Pokud je střední, tak přičteme count/2 pokud je barrier větší než 0, jinak přičteme count. Pokud je nejtěžší, tak pokud je barrier větší než 1 a count menší než 200, nepřičteme nic, pokud je barrier rovno 1 tak přičteme count/2, jinak přičteme count.

Celý předchozí odstavec tedy probíhá pro každý ze čtyř směrů zvlášť. Pak ovšem celé vyhodnocování provedeme pro dané políčko podruhé, ale s tím rozdílem, že skóre připočítáváme pro soupeřova políčka a bariéru za naše políčka. Pokud je count větší než 3, měníme jej na 15 (to je proto, že pokud se potkají dvě políčka, která jsou vedle čtveřice symbolů, je výhodnější dát svůj symbol jako pátý do řady a vyhrát než bránit). Jinak vše probíhá stejně.

Takto ohodnotíme všechna políčka. Výsledený tah se vybere jako náhodné políčko s největší hodnotou. Pokud je obtížnost nejlehčí, vybere se náhodné políčko s největší nebo druhou největší hodnotou.

# Závěr

S prací jsem spokojen. Podařilo se mi aplikaci dostat do stavu, v jakém jsem si ji představoval, i když jsem nakonec nepoužil minimaxový algoritmus umělé inteligence. Aplikace je napsána co nejlépe objektově, jak jsem to v object pascalu dokázal. Bohužel se mi nepodařilo zjistit, jak rozdělit kód do více tříd i souborů, proto je většina kódu celé hry v obslužném zdrojovém kódu hlavního okna. Myslím, že by nebylo těžké dodělat např. podporu načítání umělé inteligence z .dll knihoven nebo například hru více hráčů po síti.