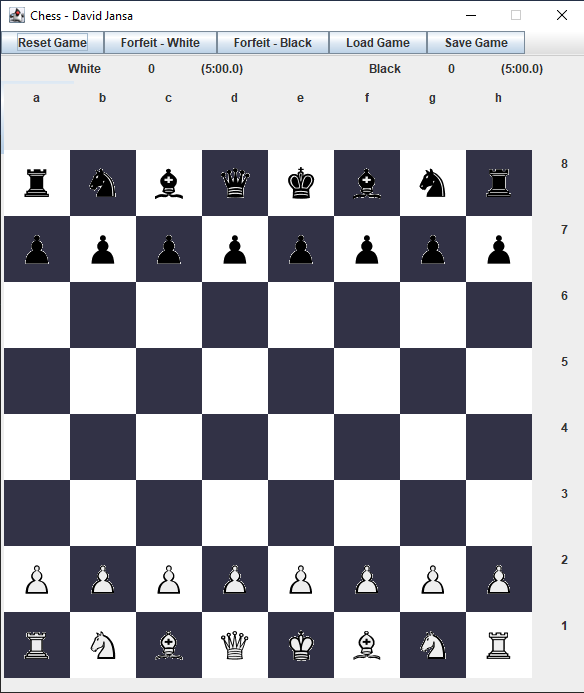
David Jansa – šachy

Návrh testovací strategie

Popis aplikace

V mém případě jsem si vybral jako testovanou aplikaci šachy. Ty jsem programoval já v rámci předmětu PJV. Hra umožňuje hraní hráč proti hráči nebo hráč proti počítači, ten je reprezentován náhodně generovanými tahy. Tato aplikace kompletně kontroluje šachová pravidla včetně speciálních tahů jako malá/velká rošáda či en passant. Zároveň je možné rozehranou partii uložit/načíst ve speciálním formátu PGN. Hra končí třemi možnostmi – šachmat, vzdání se nebo vypršení času (5 min).



Přehled částí aplikace

|  |  |  |
| --- | --- | --- |
| Proces | Funkčnost | Odpovědná třída |
| Získání informací | Před spuštěním hry uživatel zadá jméno obou hráčů a jméno souboru, pokud se bude hra ukládat. | ChessController |
| Spuštění hry | Hra se spustí se zadanými jmény, hráči (hráč/počítač) a „čerstvou“ šachovnicí | ChessController |
| Kontrola pravidel | Během hry aplikace kontroluje, jestli je provedený tah platný. | Board |
| Kontrola stavu hry | Po každém platném tahu hra zkontroluje, jestli není check/checkmate/stalemate | Board/ChessController |
| Šachové hodiny | Během celé partie hráčům ubíhá čas, když vyprší, hráč na tahu prohrál | MyTimer |
| Ukládání/Načítání partie | Rozehranou partii lze uložit/načíst ze standartního formátu PGN | PgnParser |

Prioritizace částí aplikace

Třídy rizika

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Pravděpodobnost výskytu | | | |
| MOŽNÉ POŠKOZENÍ APLIKACE V PŘÍPADĚ VÝSKYTU |  | High | Medium | Low |
| High | A | B | B |
| Medium | B | B | C |
| Low | C | C | C |

Priority

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Proces | Možné poškození | Vysvětlení | Pravděpodobnost  výskytu poškození | Vysvětlení | Třída rizika |
| Získání informací | L | Hra je funkční i bez zadání informací | L | Program prázdné informace doplňuje | C |
| Spuštení hry | H | Bez funkčního spuštění je hra nehratelná | M | Střední množství vstupních kombinací | B |
| Kontrola pravidel | H | Klíčový prvek celé aplikace | H | Složitá logika + každá úprava může ovlivnit funkčnost | A |
| Kontrola stavu hry | M | Důležité pro správné vyhodnocení vítěze | M | Hra má mnoho možných stavů a je potřeba kontrolavat velké množství věcí | B |
| Šachové hodiny | M | Důležité pro správné vyhodnocení vítěze | L | Minimální možnost k selhání | C |
| Ukládání/načítání partií | M | Samostatný, ale i důležitý prvek aplikace | H | Mnoho možných vstupů | B |

Test levels

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Proces | Třída rizika | Vývojářské testy | Systémové testy | UAT |
| Získání informací | C | nízká | nízká | střední |
| Spuštění hry | B | nízká | střední | nízká |
| Kontrola pravidel | A | vysoká | střední | nízká |
| Kontrola stavu hry | B | střední | střední | nízká |
| Šachové hodiny | C | - | nízká | - |
| Ukládání/načínání partií | B | střední | střední | nízká |

Testovací scénáře

Testy vstupů

Testována bude metoda makeMove(int start\_x, int start\_y, int dest\_x, int dest\_y) ze třídy Board. Tato metoda dostane souřadnice startovního a cílového políčka a zkontroluje, jestli je tah validní. Pokud ano, tak ho provede, přidá tah na seznam zahraných a vrátí true. V opačném případě nic nedělá a vrací false.

Políčka mohou být 4 typů:

* Prázdné políčko
* Na políčku je bílá figurka
* Na políčku je černá figurka
* Souřadnice políčka jsou mimo šachovnici

Validní tah může nabývat jen dvou hodnot

* True – tah je platný
* False – tah je neplatný

Celkově tedy máme 3 atributy – startovní políčko, cílové políčko a validní tah.

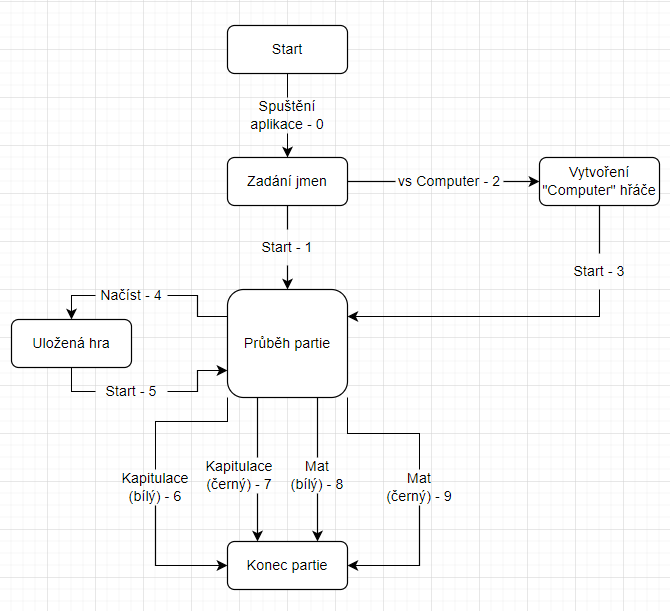
Třídy ekvivalence

|  |  |  |
| --- | --- | --- |
| Atribut | Nevalidní třída ekvivalence | Validní třída ekvivalence |
| Startovní políčko | Prázdné, mimo šachovnici | Bílá figurka, černá figurka |
| Cílové políčko | Mimo šachovnici | Bílá figurka, černá figurka, prázdné |
| Validní tah | - | True, false |

|  |  |  |
| --- | --- | --- |
| Startovní políčko | Cílové políčko | Validní tah |
| Prázdné | Prázdné | True |
| Prázdné | Bílá figurka | False |
| Prázdné | Černá figurka | True |
| Prázdné | Mimo | False |
| Bílá figurka | Bílá figurka | True |
| Bílá figurka | Černá figurka | False |
| Bílá figurka | Mimo | True |
| Bílá figurka | Prázdné | False |
| Černá figurka | Černá figurka | True |
| Černá figurka | Mimo | False |
| Černá figurka | Prázdné | True |
| Černá figurka | Bílá figurka | False |
| Mimo | Mimo | True |
| Mimo | Prázdné | False |
| Mimo | Bílá figurka | True |
| Mimo | Černá figurka | False |

Pairwise testing

Testy průchodů

Diagram průchodu jednou partií.

|  |  |  |  |
| --- | --- | --- | --- |
| Stav aplikace | Vstupy | Výstupy | Možné kombinace |
| Zadání jmen | 0 | 1, 2 | 0-1, 0-2 |
| Vytvoření „Computer“ hráče | 2 | 3 | 2-3 |
| Uložená hra | 4 | 5 | 4-5 |
| Průběh partie | 1, 3, 5 | 4, 6, 7, 8, 9 | 1-4, 1-6, 1-7, 1-8, 1-9,  3-4, 3-6, 3-7, 3-8, 3-9, 5-4, 5-6, 5-7, 5-8, 5-9 |

|  |  |
| --- | --- |
| Číslo testu | Pořadí přechodu mezi stavy |
| 1 | 0-1-4-5-6 |
| 2 | 0-2-3-6 |
| 3 | 0-1-6 |
| 4 | 0-1-7 |
| 5 | 0-1-8 |
| 6 | 0-1-9 |
| 7 | 0-2-3-4-5-6 |
| 8 | 0-2-3-7 |
| 9 | 0-2-3-8 |
| 10 | 0-2-3-9 |
| 11 | 0-1-4-5-4-6 |
| 12 | 0-1-4-5-7 |
| 13 | 0-1-4-5-8 |
| 14 | 0-1-4-5-9 |

Detailní testovací scénář

* ID: 1
* Autor: David Jansa
* Priorita: Vysoká
* Název: Fool’s mate
* Hloubka detailu: Střední
* Shrnutí: Mat v nejméně tazích pro černou stranu (5 tahů), pozitivní průchod, vyhrává bílý (strana která začínala)
* Popis: Uživatel spustí aplikaci, zvolí hru „player vs player“ a zadá jména. Dále odehraje hru za oba hráče podle předem daných pěti tahů. Hra končí vítězstvím černé (horní) strany
* Testovaci data:
  + Bílý a černý hráč – oba ovládá uživatel
  + Odehrané tahy:
    - (6, 4) -> (4, 4) – bílý pěšec na e4
    - (1, 5) -> (2, 5) – černý pěšec na f6
    - (7, 1) -> (5, 2) – bílý kůň na c3
    - (1, 6) -> (3, 6) – černý pěšec na g5
    - (7, 3) -> (3, 7) – bílá královna na h5 -> šachmat
* Očekávaný výsledek: metoda isCheckMate() vrátí true a Logger vypíše „Checkmate“ do konzole

Implementace testů

Unit testy

Cílem prvních 8 unit testů bude otestovat funkčnost metody isValidMove() u figurek královny, koně a pěšce. Tato metoda vrátí true, pokud tah odpovídá pravidlům konkrétní figurky (nebere se ohled na překážky při cestě atd.)

Queen:

* isValidMove\_diagonalMove\_trueReturned()
* isValidMove\_knightTypeMove\_falseReturned()
* isValidMove\_samePosition\_falseReturned()

Knight:

* isValidMove\_correctMove\_trueReturned()
* isValidMove\_verticalMove\_falseReturned()

Pawn:

* isValidMove\_oneTileJump\_trueReturned()
* isValidMove\_twoTileJump\_trueReturned()
* isValidMove\_threeTileJump\_falseReturned()

Dále otestujeme metodu noObstacleOnPath() ve tříde Board. Pokud je mezi startovním a cílovým polem překážka (přátelská či nepřátelská figurka) vrací true, jinak false. Těmito testy okrajově otestujeme i metodu getAllValidPath() u zvolených figurek, jelikož je zde použita. Výsledek této funkce je seznam souřadnic, přes které musí figurka jít.

* noObstacleOnPath\_noObstacle\_trueReturned()
* noObstacleOnPath\_oneObstacle\_falseReturned()

Poslední testovanou metodou bude canKingEscape() ve třídě Board. Tato metoda vrátí true, pokud je král schopný „utéct“ z pozice, na které je „checknutý“. Jinak vrací false.

* canKingEscape\_canEscape\_trueReturned()
* canKingEscape\_cannotEscape\_falseReturned()

Testy s mockito

Tyto testy budou probíhat také ve tříde BoardTest

* isNotOutOfRange\_outOfRangeButValid\_trueReturned()
* isNotOutOfRange\_validMoveButOutOfRange\_falseReturned()
* verify\_setPieceInInitializingBoard\_invokedEightTimes\_trueReturned()
* getPiece\_getPiecesWithSameType\_trueReturned()
* getPiece\_getPawnCanJumpTwoTiles\_trueReturned()

Procesní testy

Procesní testy testují předem vymyšlené scénáře. Nacházejí se ve třídě „GameTest“.

* processTest\_foolsMate\_whiteWins()
* processTest\_whitePlays\_rightCastlingMove()
* processTest\_whiteAttacks\_enPassantBlackPawn()
* processTest\_whiteChecksBlackKing\_blackBlockAndCapture()