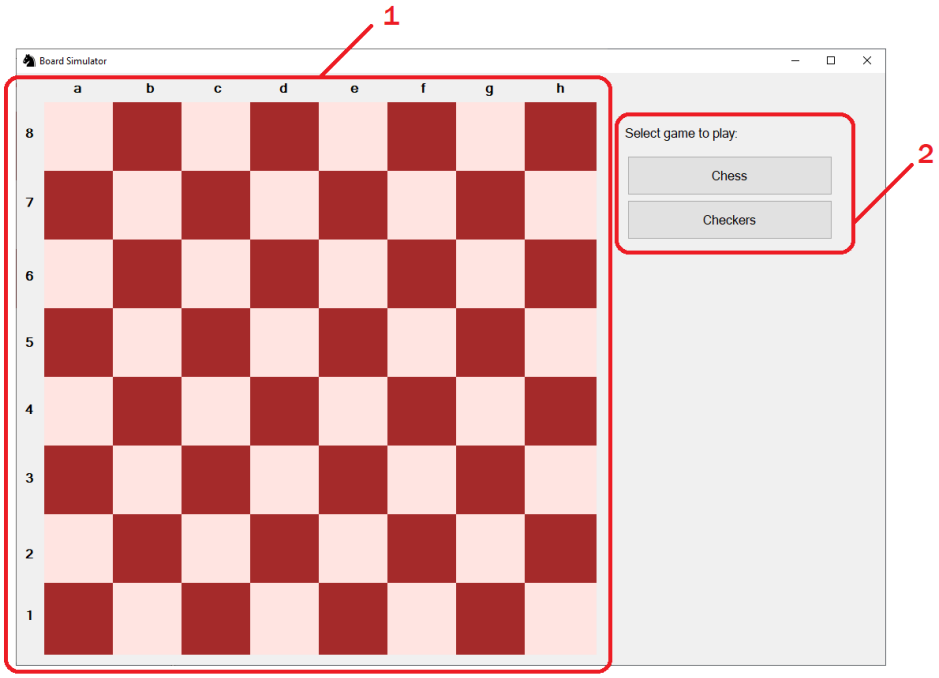
Chess Board Simulator

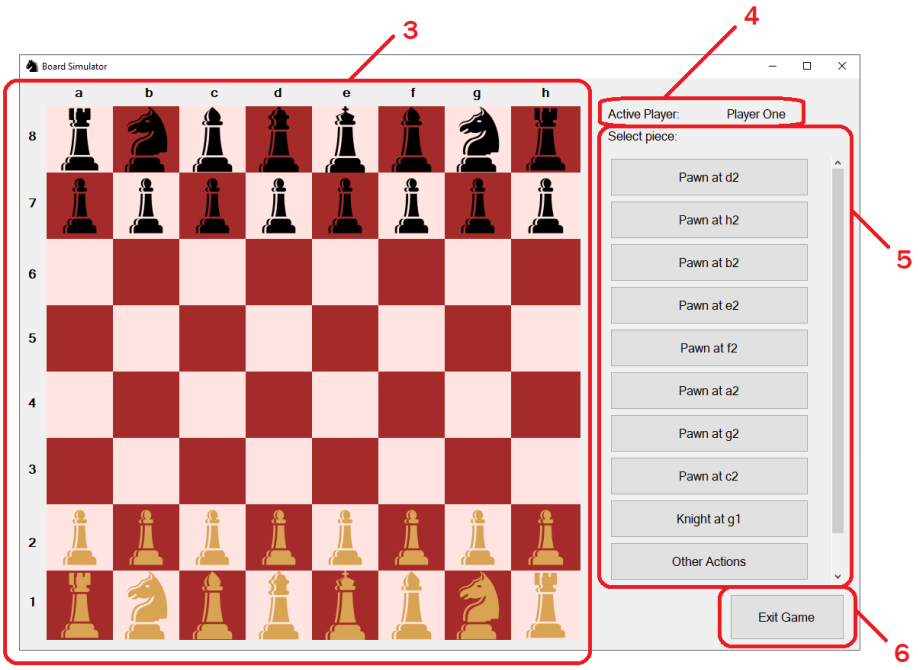
Dokumentacja Użytkowa

Dokumentacja użytkowa dla graczy:

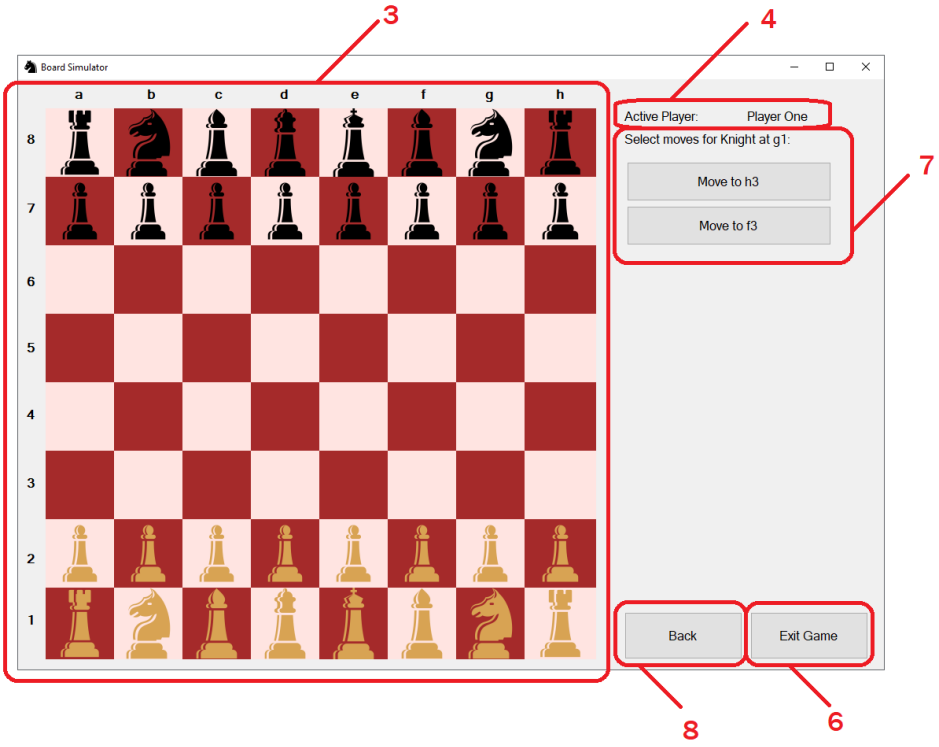
Ekran Powitalny:



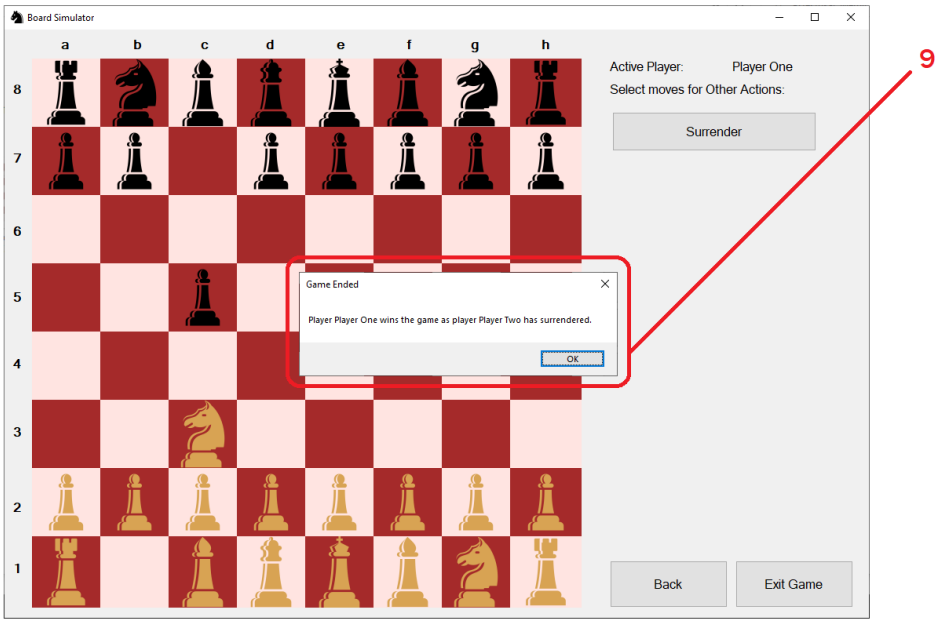
Ekran Wyboru Pionka:



Ekran Wyboru Ruchu Pionka:



Ekran Końca Gry:



Legenda:

1. Wizualizacja pustej planszy.
2. Lista dostępnych gier, wybranie gry z listy rozpoczyna grę.
3. Wizualizacja wybranej gry na planszy.
4. Informacja o aktywnym graczu.
5. Lista pionków które mogą się poruszyć, wybranie pionka przenosi nas do ekranu wyboru ruchu pionka.
6. Przycisk przerywający grę wcześnie i przywracający ekran powitalny.
7. Informacja o aktualnie wybranym pionku i lista jego możliwych ruchów, wybranie ruchu z listy wykonuje ten ruch i rozpoczyna turę kolejnego gracza.
8. Przycisk powrotu do listy pionków.
9. Wiadomość informująca o zakończeniu gry i jej wyniku.

Dokumentacja użytkowa dla deweloperów:

Aby dodać nową grę do silnika należy zaimplementować klasy dziedziczące po klasach “Game” i “Pawn” reprezentujące odpowiednio implementowaną grę i pionki używane w tej grze (jedna gra może i powinna używać wiele pionków). Na końcu należy dodać wskaźnik do naszej implementacji klasy “Game” do zmiennej “allGames” w funkcji “main” klasy “MainForm” aby aplikacja mogła wyświetlić nowo dodaną grę.

Symulacja gry odbywa się w turach wykonywanych przez dwóch graczy na przemian. Struktura pojedynczej tury wygląda następująco:

1. Obliczenie przez pionki aktywnego gracza wszystkich swoich możliwych ruchó metodą getMoves()
2. Wykonanie metody PostCalculationUpdates() pozwalającej na modyfikację obliczonych ruchów z perspektywy gry
3. Wykonanie metody CheckGameEndConditions() sprawdzającej czy warunki zakończenia gry już zostały spełnione, zakończenie gry, jeżeli tak jest.
4. Wyświetlenie dla aktywnego gracza listy możliwych dla niego ruchów.
5. Wybranie przez aktywnego gracza jednego z ruchów i wykonanie go na planszy.
6. Wykonanie metody PostMoveUpdates() pozwalające na dodatkowe modyfikacje planszy z poziomu gry zgodnie z jej logiką.
7. Rotacja pozycji aktywnego gracza.
8. Powrót do punktu 1.

Poprawna implementacja klasy “Game” powinna nadpisywać następujące metody:

1. Konstruktor - należy wywołać konstruktor klasy bazowej i nadpisać pole “name” wybraną nazwą gry, nazwa będzie wyświetlana na liście gier.
2. void setupBord() - metoda wywoływana przy starcie (lub restarcie) nowej gry, należy zainicjalizować w zmienne: “gameboard”, “gameStatus”, “activePlayer” i wywołać metodę “CalculateAllPosibleMoves” dla aktywnego gracza. Przy inicjalizacji “gameboard” należy ustawić pionki przypisane implementowanej grze na właściwe im miejsca.
3. void CheckGameEndConditions() - metoda sprawdzająca czy gra została zakończona, uruchamiana przed ruchem aktywnego gracza, jeżeli warunki kończące grę zostały spełnione

należy ustawić zmienną “gameStatus.first” na prawdę a “gameStatus.second” na wiadomość jaka powinna się wyświetlić użytkownikowi, informująca go o wyniku gry.

4. void PostCalculationUpdates() - metoda uruchamiana po tym jak pionki obliczą swoje możliwe ruchy, pozwalająca na edycje zmiennej “rememberedMovesMap”, dodając nowe ruchy lub usuwając już dodane, z poziomu gry a nie pionka, zgodnie z logiką implementowanej gry. Implementacja tej metody jest opcjonalna.
5. void PostMoveUpdates() - metoda uruchamiana po ruchu aktywnego gracza, pozwalająca na dodatkowe modyfikacje planszy z poziomu gry, zgodnie z logiką implementowanej gry. Implementacja tej metody jest opcjonalna.

Poprawna implementacja klasy “Piece” powinna nadpisywać następujące metody:

1. Konstruktor – powinien wywoływać konstruktor klasy bazowej, nadpisujący we własny sposób pola “name, “loyalty” i “icon”.
2. Bishop* clone() - prosta metoda zwracająca głęboką kopię tej samej klasy, którą implementujemy.
3. vector<BoardMove*> getMoves() - metoda wywoływana na potrzeby ustalenia możliwych ruchów wybranego pionka na planszy. Powinna zwracać vector wskaźników do klas BoardMove opisujących każdy możliwy pojedynczy ruch pionka na planszy. Same ruchy będą zależeć od natury implementowanej gry i pionka. Ruch może zawierać dowolną liczbę modyfikacji planszy.