bool is\_valid\_action(Player \*p, GameState \*g, ActionType action) {

    if (p->in\_game != PLAYING || p->status != STATUS\_ACTIVE || p->credits <= 0) {

        return false;

    }

    int call\_amount = g->current\_bet - p->current\_bet;

    switch (action) {

        case ACTION\_CHECK:

            return call\_amount == 0;

        case ACTION\_CALL:

            return call\_amount > 0 && p->credits >= call\_amount;

        case ACTION\_BET:

            return g->current\_bet == 0 && p->credits > 0;

        case ACTION\_RAISE: {

            int max\_possible\_raise\_amount = p->credits - call\_amount;

            return g->current\_bet > 0 && max\_possible\_raise\_amount >= g->last\_raise\_amount;

        }

        case ACTION\_ALL\_IN:{

            if (p->credits <= 0) return false;

            if (g->current\_bet == 0) {

                // All-in as a bet

                return true;

            } else if (p->credits + p->current\_bet > g->current\_bet) {

                // All-in as a raise

                return true;

            } else if (p->credits + p->current\_bet == g->current\_bet) {

                // All-in as a call

                return true;

            } else {

                // All-in is less than call — still allowed, but not re-raisable

                return true;

            }

        }

        case ACTION\_FOLD:

            return p->status == STATUS\_ACTIVE;

        default:

            return false;

    }

}

|  |  |
| --- | --- |
| #ifndef IN\_GAME\_H  #define IN\_GAME\_H  typedef enum playerInGame {      QUIT = 0,      PLAYING = 1  } InGame;  #endif // IN\_GAME\_H | #ifndef PLAYER\_STATUS\_H  #define PLAYER\_STATUS\_H  typedef enum playerStatus{      STATUS\_OUT = 0,     // Quit or out of credits      STATUS\_ACTIVE = 1,  // Currently in the hand      STATUS\_FOLDED = 2   // Folded during this hand  } PlayerStatus;  typedef enum {      ACTION\_CHECK,      ACTION\_CALL,      ACTION\_BET,      ACTION\_RAISE,      ACTION\_ALL\_IN,      ACTION\_FOLD  } ActionType;  #endif // PLAYER\_STATUS\_H |
| #ifndef GAMESTATE\_H  #define GAMESTATE\_H  extern int last\_raise\_amount;  // size of last raise made this round  typedef enum {      STAGE\_PREFLOP,      STAGE\_FLOP,      STAGE\_TURN,      STAGE\_RIVER,      STAGE\_SHOWDOWN  } RoundStage;  typedef struct {      RoundStage stage;      int pot;      int dealer\_index;      int current\_bet;      int small\_blind;      int big\_blind;      int last\_raise\_amount; // Minimum raise required  } GameState;  #endif // GAMESTATE\_H |

#include "../src/player.h"

#include "../src/game.h"

#include "../src/gamestate.h"

#include "../src/playerStatus.h"

#include <assert.h>

#include <stdio.h>

// ACTION\_CHECK

void test\_action\_check() {

    // ✅ Case: No current bet

    Player anna = {

        .credits = 500,

        .current\_bet = 0,

        .in\_game = PLAYING,

        .status = STATUS\_ACTIVE

    };

    GameState game\_state = {

        .current\_bet = 0

    };

    printf("Player: Anna\n");

    printf("credits: %i\n",anna.credits);

    printf("her current bet: %i\n",anna.current\_bet);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("in the game: %s\n",anna.in\_game == PLAYING ? "yes":"no");

    printf("in the hand: %s\n",anna.status == STATUS\_ACTIVE ? "yes":"no");

    printf("Can Anna Check ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_CHECK) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_CHECK) == true);

    // ❌ Case: There's a bet to match

    game\_state.current\_bet = 50; // Now there's a bet on the table

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_CHECK) == false);

    // ⚠️ Edge: Player already matched bet exactly (call\_amount == 0)

    anna.current\_bet = 50;

    printf("Anna's current bet: %i\n",anna.current\_bet);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

     printf("Can Anna Check ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_CHECK) ? "yes":"no");

     assert(is\_valid\_action(&anna, &game\_state, ACTION\_CHECK) == true);

}

// ACTION\_CALL

void test\_action\_call() {

    // ✅ Case: call\_amount > 0, player has enough credits

    Player anna = {

        .credits = 500,

        .current\_bet = 0,

        .in\_game = PLAYING,

        .status = STATUS\_ACTIVE

    };

    GameState game\_state = {

        .current\_bet = 55

    };

    printf("Player: Anna\n");

    printf("credits: %i\n",anna.credits);

    printf("her current bet: %i\n",anna.current\_bet);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("in the game: %s\n",anna.in\_game == PLAYING ? "yes":"no");

    printf("in the hand: %s\n",anna.status == STATUS\_ACTIVE ? "yes":"no");

    printf("Can Anna Place a bet ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_CALL) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_CALL) == true);

    // ❌ Case: call\_amount > 0, not enough credits

    game\_state.current\_bet = 501;

    printf("Anna's credits: %i\n",anna.credits);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("Can Anna CALL ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_CALL) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_CALL) == false);

    // ⚠️ Edge: call\_amount == player's credits exactly (Call = All-in)

    game\_state.current\_bet = 500;

    anna.current\_bet = 500;

    printf("Anna's credits: %i\n",anna.credits);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("Can Anna CALL ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_CALL) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_CALL) == false);

}

// ACTION\_BET

void test\_action\_bet() {

    // ✅ Case: current\_bet == 0, player has credits

    Player anna = {

        .credits = 500,

        .current\_bet = 0,

        .in\_game = PLAYING,

        .status = STATUS\_ACTIVE

    };

    GameState game\_state = {

        .current\_bet = 0

    };

    printf("Player: Anna\n");

    printf("credits: %i\n",anna.credits);

    printf("her current bet: %i\n",anna.current\_bet);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("in the game: %s\n",anna.in\_game == PLAYING ? "yes":"no");

    printf("in the hand: %s\n",anna.status == STATUS\_ACTIVE ? "yes":"no");

    printf("Can Anna Place a bet ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_BET) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_BET) == true);

    // ❌ Case: current\_bet > 0 (someone already bet)

    game\_state.current\_bet = 55;

    printf("Player: Anna\n");

    printf("credits: %i\n",anna.credits);

    printf("her current bet: %i\n",anna.current\_bet);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("in the game: %s\n",anna.in\_game == PLAYING ? "yes":"no");

    printf("in the hand: %s\n",anna.status == STATUS\_ACTIVE ? "yes":"no");

    printf("Can Anna Place a bet ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_BET) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_BET) == false);

    printf("No she can call,raise or fold\n");

    // ⚠️ Edge: credits == 1 (minimum bet possible)

    game\_state.current\_bet = 0;

    anna.credits = 1;

    printf("Player: Anna\n");

    printf("credits: %i\n",anna.credits);

    printf("her current bet: %i\n",anna.current\_bet);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("in the game: %s\n",anna.in\_game == PLAYING ? "yes":"no");

    printf("in the hand: %s\n",anna.status == STATUS\_ACTIVE ? "yes":"no");

    printf("Can Anna Place a bet ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_BET) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_BET) == true);

}

// ACTION\_RAISE

void test\_action\_raise() {

    // ✅ Case: current\_bet > 0, player has enough for call + raise

    Player anna = {

        .credits = 500,

        .current\_bet = 0,

        .in\_game = PLAYING,

        .status = STATUS\_ACTIVE

    };

    GameState game\_state = {

        .current\_bet = 99

    };

    printf("Player: Anna\n");

    printf("credits: %i\n",anna.credits);

    printf("her current bet: %i\n",anna.current\_bet);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("in the game: %s\n",anna.in\_game == PLAYING ? "yes":"no");

    printf("in the hand: %s\n",anna.status == STATUS\_ACTIVE ? "yes":"no");

    printf("Can Anna Call ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_CALL) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_RAISE) == true);

    printf("Can Anna Raise ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_RAISE) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_RAISE) == true);

    // ❌ Case: player has just enough to call but not raise

    game\_state.current\_bet = 500;

    game\_state.last\_raise\_amount = 20;

    printf("credits: %i\n",anna.credits);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("Can Anna Call ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_CALL) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_CALL) == true);

    printf("Can Anna Raise ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_RAISE) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_RAISE) == false);

    // ⚠️ Edge: player raise = all-in but doesn’t meet minimum raise

    game\_state.last\_raise\_amount = 50;

    game\_state.current\_bet = 500;

    anna.credits=500;

    printf("Anna's credits: %i\n",anna.credits);

    printf("Minimum Raise: %i\n", game\_state.last\_raise\_amount);

    printf("Can Anna Raise ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_RAISE) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_RAISE) == false);

     printf("No She can only go all-in\n");

}

// ACTION\_ALL\_IN

void test\_action\_all\_in() {

    // ✅ Case: all-in used as a bet (current\_bet == 0)

    Player anna = {

        .credits = 500,

        .current\_bet = 0,

        .in\_game = PLAYING,

        .status = STATUS\_ACTIVE

    };

    GameState game\_state = {

        .current\_bet = 0

    };

    printf("Player: Anna\n");

    printf("credits: %i\n",anna.credits);

    printf("her current bet: %i\n",anna.current\_bet);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("in the game: %s\n",anna.in\_game == PLAYING ? "yes":"no");

    printf("in the hand: %s\n",anna.status == STATUS\_ACTIVE ? "yes":"no");

    printf("Can Anna Go ALL-IN as a bet ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_ALL\_IN) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_ALL\_IN) == true);

    // ✅ Case: all-in used to call exactly

    game\_state.current\_bet = 500;

    printf("Player: Anna\n");

    printf("credits: %i\n",anna.credits);

    printf("her current bet: %i\n",anna.current\_bet);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("in the game: %s\n",anna.in\_game == PLAYING ? "yes":"no");

    printf("in the hand: %s\n",anna.status == STATUS\_ACTIVE ? "yes":"no");

    printf("Can Anna Go ALL-IN as a call ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_ALL\_IN) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_ALL\_IN) == true);

    // ⚠️ Case: all-in < call (can’t raise, but still allowed)

}

// ACTION\_FOLD

void test\_action\_fold() {

    // ✅ Case: Player is active

        Player anna = {

        .credits = 500,

        .current\_bet = 0,

        .in\_game = PLAYING,

        .status = STATUS\_ACTIVE

    };

    GameState game\_state = {

        .current\_bet = 0

    };

    printf("Player: Anna\n");

    printf("credits: %i\n",anna.credits);

    printf("her current bet: %i\n",anna.current\_bet);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("in the game: %s\n",anna.in\_game == PLAYING ? "yes":"no");

    printf("in the hand: %s\n",anna.status == STATUS\_ACTIVE ? "yes":"no");

    printf("Can FOLD ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_FOLD) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_FOLD) == true);

    // ❌ Case: Player already folded or out

    anna.status = STATUS\_OUT;

    printf("Player: Anna\n");

    printf("credits: %i\n",anna.credits);

    printf("her current bet: %i\n",anna.current\_bet);

    printf("latest bet in hand: %i\n",game\_state.current\_bet);

    printf("in the game: %s\n",anna.in\_game == PLAYING ? "yes":"no");

    printf("in the hand: %s\n",anna.status == STATUS\_ACTIVE ? "yes":"no");

    printf("Can Anna FOLD ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_FOLD) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_FOLD) == false);

    printf("NOW:\n");

    anna.status = STATUS\_FOLDED;

    printf("in the game: %s\n",anna.in\_game == PLAYING ? "yes":"no");

    printf("in the hand: %s\n",anna.status == STATUS\_ACTIVE ? "yes":"no");

    printf("Can Anna FOLD ? %s\n",is\_valid\_action(&anna, &game\_state, ACTION\_FOLD) ? "yes":"no");

    assert(is\_valid\_action(&anna, &game\_state, ACTION\_FOLD) == false);

}

// Main

int main() {

    printf("START============ TESTING - CHECK ============\n");

    test\_action\_check();

    printf("END============ TESTING - CHECK ============\n\n");

    printf("START============ TESTING - CALL ============\n");

    test\_action\_call();

    printf("END============ TESTING - CALL ============\n\n");

    printf("START============ TESTING - BET ============\n");

    test\_action\_bet();

    printf("END============ TESTING - BET ============\n\n");

    printf("START============ TESTING - RAISE ============\n");

    test\_action\_raise();

    printf("END============ TESTING - RAISE ============\n\n");

    printf("START============ TESTING - ALL\_IN ============\n");

    test\_action\_all\_in();

    printf("END============ TESTING - ALL\_IN ============\n\n");

    printf("START============ TESTING - FOLD ============\n");

    test\_action\_fold();

    printf("END============ TESTING - FOLD ============\n\n");

    printf("All tests passed.\n");

    return 0;

}