1. Karta

```
struct Card {
    enum Element { Fire, Water, Earth, Air };
    Element element;
    int value;
};
```

2. Pole mriežky (GridTile)

Na mriežke sú tvary, ktoré sa obsadzujú kartou.

```
struct GridTile {
    bool occupied = false;
    int owner = -1;        // 0 = hráč 1, 1 = hráč 2
    Card placedCard;       // karta použitá na obsadenie
};
```

3. Tvar na mriežke (ShapeType)

```
enum class ShapeType {
    Cross,
    S-shape,
    Square,
    L-shape
};
```

4. Mriežka

```
class Grid {
public:
    int width, height;
    vector<vector<GridTile>> tiles; // 2-D matica na reprezentaciu herného
polia

    Grid(int w, int h) : width(w), height(h), tiles(h,
vector<GridTile>(w)) {}

    // Kontrola hranic
    bool isInside(int x, int y) const {
        return x >= 0 && x < width && y >= 0 && y < height;
    }

    bool placeCard(int x, int y, const Card& card, int playerId) {
        if (!isInside(x, y)) return false;
        if (tiles[y][x].occupied) return false;

        tiles[y][x].occupied = true;
        tiles[y][x].owner = playerId;
        tiles[y][x].placedCard = card;
        return true;
    }

    Card removeCard(int x, int y) {
        Card c = tiles[y][x].placedCard;
        tiles[y][x].occupied = false;
        tiles[y][x].owner = -1;
        return c;
    }
```

```
};
```

5. Hráč (Player)
Každý hráč má:
- 3 karty v ruke
- svoj balíček ~10 kariet
- zozbierané ("vyhrané") karty
- ID hráča

```
class Player {
public:
    int id;                          // 0 alebo 1
    vector<Card> hand;          // 3 karty
    vector<Card> deck;          // vlastný balíček
    vector<Card> wonCards;      // získané karty z mriežky

    Player(int playerId) : id(playerId) {}

    // Potiahnutie kariet zo svojho balíčka
    void drawFromDeck(int n) {
        for (int i = 0; i < n && !deck.empty(); i++) {
            hand.push_back(deck.back());
            deck.pop_back();
        }
    }

    // Vyhodenie všetkých kariet z ruky
    void discardAllFromHand(vector<Card>& discardHand) {
        for (auto& c : hand) discardHand.push_back(c);
        hand.clear();
    }
};
```

6. Veľký balíček (BigDeck)
Používa sa pri situácii, že hráč nemá žiadnu hrateľnú kartu.

```
class BigDeck {
public:
    vector<Card> cards;
    vector<Card> draw(int n) {
        vector<Card> drawn;
        for (int i = 0; i < n && !cards.empty(); i++) {
            drawn.push_back(cards.back());
            cards.pop_back();
        }
        return drawn;
    }

    bool isEmpty() const {
        return cards.empty();
    }
};
```

7. Discard pre karty odhodené z ruky

```
vector<Card> discardHand;
```

8. Kontrola, či hráč môže zahrať

```
bool canPlayerPlay(const Player& p, const Player& opponent, const Grid&
grid) {
    // zatiaľ len placeholder, reálna logika podľa hodnot a susedov
    return true;
}
```

9. Situácia: hráč nemôže zahrať – odhodenie ruky + potiahnutie z veľkého balíčka

```
void handleNoPlayableCards(Player& player, BigDeck& bigDeck,
std::vector<Card>& discardHand) {
    // 1. Odhodiť celú ruku
    player.discardAllFromHand(discardHand);

    // 2. Potiahnuť 3 nové karty z veľkého balíka
    std::vector<Card> newCards = bigDeck.draw(3);
    for (auto& c : newCards) {
        player.hand.push_back(c);
    }
}
```

10. Riešenie výsledka súboja na mriežke

```
void resolveCombatAndReplace(Grid& grid, int x, int y,
                             const Card& winningCard,
                             int winningPlayer,
                             std::vector<Card>& discardPile)
{
    // 1. Zober porazenú kartu z mriežky
    Card defeated = grid.removeCard(x, y);

    // 2. Ulož ju do discard pile (nie k víťazovi!)
    discardPile.push_back(defeated);

    // 3. Umiestni víťazovu kartu na pole
    grid.placeCard(x, y, winningCard, winningPlayer);
}
```

11. Hlavná trieda Game – ktorá všetko spája

```
class Game {
public:
    Grid grid;
    Player player1;
    Player player2;
    BigDeck bigDeck;
    vector<Card> discardHand;

    int currentPlayer = 0;

    Game(int w, int h)
        : grid(w, h), player1(0), player2(1) {}

    Player& getCurrentPlayer() {
        return currentPlayer == 0 ? player1 : player2;
    }
```

```
    Player& getOpponent() {
        return currentPlayer == 0 ? player2 : player1;
    }

    void endTurn() {
        currentPlayer = 1 - currentPlayer;
    }
};
```