

GAME MAIN

```
graph LR; START([START]) --> MAKE_BOARD[board = MAKE_BOARD (WIDTH, HEIGHT)]; MAKE_BOARD --> DISPLAY_GAME_INTRO[DISPLAY_GAME_INTRO]; DISPLAY_GAME_INTRO --> MAKE_CHARACTER[MAKE_CHARACTER]; MAKE_CHARACTER --> DISPLAY_LOCATION[DISPLAY_LOCATION (board, character)]; DISPLAY_LOCATION --> DEFINE_STATUS[define status = 'game ongoing']; DEFINE_STATUS --> IS_STATUS_GAME_ONGOING{is status 'game ongoing?'}; IS_STATUS_GAME_ONGOING -- True --> GET_USER_CHOICE[user_choice = GET_USER_CHOICE (user move options)]; GET_USER_CHOICE --> IS_USER_CHOICE_EQUAL_TO_QUIT{is user_choice equal to 'quit'?}; IS_USER_CHOICE_EQUAL_TO_QUIT -- YES --> STATUS_QUIT[status = 'quit']; STATUS_QUIT --> IS_STATUS_GAME_ONGOING; IS_USER_CHOICE_EQUAL_TO_QUIT -- NO --> VALID_MOVE[valid_move = VALIDATE_MOVE (user_choice, character)]; VALID_MOVE --> IS_VALID_MOVE_EQUAL_TO_TRUE{is valid_move equal to 'True'?}; IS_VALID_MOVE_EQUAL_TO_TRUE -- YES --> CHARACTER_EXPLORE[status = CHARACTER_EXPLORE (user_choice, character, board)]; CHARACTER_EXPLORE --> IS_STATUS_GAME_ONGOING; IS_VALID_MOVE_EQUAL_TO_TRUE -- NO --> PRINT_MESSAGE[/print 'You can't move in this direction'/]; PRINT_MESSAGE --> DISPLAY_LOCATION; DISPLAY_LOCATION --> GAME_ENDING_MESSAGE[GAME_ENDING_MESSAGE (status)]; GAME_ENDING_MESSAGE --> END([END]);
```

The flowchart illustrates the main game loop. It begins with a 'START' terminal, followed by initialization steps: 'board = MAKE_BOARD (WIDTH, HEIGHT)', 'DISPLAY_GAME_INTRO', 'MAKE_CHARACTER', and 'DISPLAY_LOCATION (board, character)'. The status is then defined as 'game ongoing'. A decision diamond checks 'is status "game ongoing?"'. If 'True', it proceeds to 'user_choice = GET_USER_CHOICE (user move options)'. Another decision diamond checks 'is user_choice equal to "quit?"'. If 'YES', the status is set to 'quit' and the loop returns to the 'is status "game ongoing?"' check. If 'NO', it proceeds to 'valid_move = VALIDATE_MOVE (user_choice, character)'. A third decision diamond checks 'is valid_move equal to "True"?'. If 'YES', the status is set to 'CHARACTER_EXPLORE (user_choice, character, board)' and the loop returns to the 'is status "game ongoing?"' check. If 'NO', a message is printed: 'You can't move in this direction', and the flow returns to 'DISPLAY_LOCATION (board, character)'. Finally, the 'GAME_ENDING_MESSAGE (status)' is displayed, and the process ends at the 'END' terminal.

CHARACTER EXPLORE

```

graph LR
    Start([CHARACTER_EXPLORE  
user_choice, character]) --> Move[MOVE_CHARACTER  
(user_choice, character)]
    Move --> Display[DISPLAY_LOCATION  
(board, character)]
    Display --> Define[define status  
(game_engine)]
    Define --> CheckBoss[CHECK_FOR_BOSS(character)]
    CheckBoss --> IsBoss{is CHECK_FOR_BOSS  
True?}
    IsBoss -- YES --> FightBoss[status = COMBAT_WITH_FINAL_BOSS(character)]
    IsBoss -- NO --> CheckFoe[CHECK_FOR_FOE(character)]
    CheckFoe --> IsFoe{is CHECK_FOR_FOE  
True?}
    IsFoe -- YES --> CreateFoe[foo = CREATE_FOE(character)]
    CreateFoe --> GetChoice[user_choice = GET_USER_CHOICE  
(user_choice)]
    GetChoice --> IsFight{is user_choice = Fight?}
    IsFight -- YES --> Combat[COMBAT  
(character, foe)]
    IsFight -- NO --> Files[CHARACTER_FILES  
(character)]
    FightBoss --> Return([RETURN  
status])
    Combat --> CheckStatus[status = CHECK_if_CHARACTER_DEAD(character)]
    Files --> CheckStatus
    CheckStatus --> Return
  
```

The flowchart for 'CHARACTER EXPLORE' starts with an oval 'CHARACTER_EXPLORE' containing 'user_choice, character'. It proceeds through several rectangular process blocks: 'MOVE_CHARACTER (user_choice, character)', 'DISPLAY_LOCATION (board, character)', 'define status (game_engine)', 'CHECK_FOR_BOSS(character)', and 'CHECK_FOR_FOE(character)'. Decision diamonds follow: 'is CHECK_FOR_BOSS True?' (YES leads to 'status = COMBAT_WITH_FINAL_BOSS(character)', NO leads to 'CHECK_FOR_FOE(character)'); 'is CHECK_FOR_FOE True?' (YES leads to 'foo = CREATE_FOE(character)', NO leads to 'CHARACTER_FILES(character)'); 'user_choice = GET_USER_CHOICE(user_choice)' leads to 'is user_choice = Fight?'; 'is user_choice = Fight?' (YES leads to 'COMBAT(character, foe)', NO leads to 'CHARACTER_FILES(character)'). A final 'status = CHECK_if_CHARACTER_DEAD(character)' block receives input from 'COMBAT' and 'CHARACTER_FILES'. The process ends at a green oval 'RETURN status'.

The figure contains three flowcharts for the game's logic:

- INTRO & OUTRO:** Starts with 'GAME_ENDING_MESSAGE (initial)', leading to a decision 'is status user choice out of 5'. If 'YES', it prints 'ending message that user decided to quit' and 'GOAL_ACHIEVED_MESSAGES', then 'print ending ascii art' and 'END'. If 'NO', it leads to another decision 'is status home out of 5'. If 'YES', it prints 'ending message that user's character died' and 'GOAL_ACHIEVED_MESSAGES', then 'END'. If 'NO', it prints 'game intro to user' and 'END'.
- CHECK STATUS:** Starts with 'CHECK_FOR_BOSS(character)', leading to a decision 'are character's coordinates == 24 and coordinate == 10'. If 'YES', it returns 'TRUE'. If 'NO', it returns 'FALSE'. Then, it goes to 'CHECK_FOR_FOE()', which generates 'True or False randomly with 20% chance of True and 80% chance of False', and returns 'True or False'. Next is 'CHECK_FOE_RUN_AWAY()', which also generates 'True or False randomly with 20% chance of True and 80% chance of False', and returns 'True or False'. Finally, it goes to 'CHECK_IF_CHARACTER_DEAD()', which checks 'is character's hp less than or equal to zero'. If 'YES', it returns 'character_dead'. If 'NO', it returns 'character_alive'.
- CHECK LEVEL UP!:** Starts with 'CHECK_FOR_LEVEL_UP()', leading to a decision 'is character level 3?'. If 'YES', it goes to 'END'. If 'NO', it checks 'level < transferLevel'. If 'YES', it prints 'level message which let the user know how much xp he earned and current xp'. Then, it checks 'is character's level > and character's hp > bigger than or equal to level 2 requires xp?'. If 'YES', it checks 'character level > 3, max hp <= 10, reset current hp to max hp'. If 'YES', it checks 'character level > 3, max hp <= 15, reset current hp to max hp'. Both 'YES' paths lead to 'print message that let the user know his current level is up and class name is changed', then 'display_character_stats', and finally 'END'. The 'NO' path from the first decision also leads to 'END'.

```

graph TD
    L1[get description randomly from L1<br/>REGION DESCRIPTION which is<br/>a tuple] -- Yes --> L1_Valid{is x coord and y coord smaller than<br/>20?}
    L1 -- No --> L1
    L2[get description randomly from L2<br/>REGION DESCRIPTION which is<br/>a tuple] -- Yes --> L2_Valid{is x coord and y coord smaller than<br/>20?}
    L2 -- No --> L2
    L1_Valid -- Yes --> L2_Valid
    L1_Valid -- No --> L1
    L2_Valid -- Yes --> L2_Invalid{get description randomly from L2<br/>REGION DESCRIPTION which is<br/>a tuple}
    L2_Valid -- No --> L2
    L2_Invalid --> Return1[RETURN<br/>Region1]
    L2_Valid -- Yes --> AskUser[/Ask user for<br/>(name for character)/]
    AskUser --> DisplayClass[DISPLAY CHARACTER<br/>(CLASS) INFORMATION]
    DisplayClass --> SetStats[SET UP CHARACTER<br/>(STATS_BY_CLASS)]
    SetStats --> PrintChar[/print<br/>"character created"<br/>to user/]
    PrintChar --> ReturnChar[RETURN<br/>character]
    ReturnChar --> End1[END]
    SetStats --> End2[END]
    
```

The flowchart illustrates the character creation process. It begins with a loop to generate random region descriptions from L1 and L2 regions until a valid one is found. Then, it asks the user for a character name, displays character class information, sets up character stats, and prints the created character. The process ends with a return statement.

```

graph TD
    subgraph CHARACTER_CREATION [CHARACTER CREATION]
        C1[CREATE_FOE(character)] --> C1a[create three dictionaries containing numbers and assign foe's name using enemies and a list of foe names. (current, boss, weak)]
        C1a --> C1b[assign randomly one of the three different foes to both names of three dictionaries]
        C1b --> C1c[According to character's region and ask choice to both names of that area come out]
        C1c --> C1d{print character's name and ask choice next action}
        C1d --> C1e[RETURN foe]
    end

    subgraph CHARACTER_LOCATION [CHARACTER LOCATION]
        C2[CREATE_BOSS(character)] --> C2a[look & discover which contains foe hp, max hp, exp, and max exp of boss]
        C2a --> C2b[/print alert message that user cannot run away/]
        C2b --> C2c[RETURN boss]
    end

    subgraph CHARACTER_INFO [CHARACTER & FOE INFO]
        C3[DISPLAY_CHARACTER_STAT(character)] --> C3a[/print character's level, max hp, exp/]
        C3a --> C3b[character_damage_points(character)]
        C3b --> C3c[/print character's same ability according to each class/]
        C3c --> C3d[END]

        C4[DISPLAY_LOCATION(boss, character)] --> C4a[character's x and y coordinate  
1. smaller than 10 <= region <= level 1 area  
2. smaller than 20 <= region <= level 2 area  
3. smaller than 30 <= region <= level 3 area]
        C4a --> C4b[/print region of current line and current coordinate/]
        C4b --> C4c[display_mini_map(character)]
        C4c --> C4d[END]

        C5[DISPLAY_MINI_MAP(character)] --> C5a[location = (character x coordinate, character y coordinate)]
    end

    subgraph GAME_LOOP [GAME LOOP]
        C6[DISPLAY_CHARACTER_HP(character)] --> C6a{is foe hp = 0?}
        C6a -- YES --> C6b[/current hp = value character key "hp"/]
        C6b --> C6c[/print "foe is equal to or smaller than zero, stop"]/
        C6c --> C6d[END]

        C6a -- NO --> C7[DISPLAY_CHARACTER_MP(character)] --> C7a{is character mp > 0?}
        C7a -- YES --> C7b[/print character's current mp "foe is equal to or smaller than zero, stop"/]
        C7b --> C7c[END]

        C7a -- NO --> C8{is x coordinate, y, 5 = row and y coordinate, 5 = column?}
        C8 -- YES --> C8a[/print [B]/]
        C8 -- NO --> C9{is x coordinate, y, 5 = row and y coordinate, 5 = column?}
        C9 -- YES --> C9a[/print [B]/]
        C9 -- NO --> C10{is x coordinate, y, 5 = row and y coordinate, 5 = column?}
        C10 -- YES --> C10a[/print [B]/]
        C10 -- NO --> C11{is x coordinate, y, 5 = row and y coordinate, 5 = column?}
        C11 -- YES --> C11a[/print [B]/]
        C11 -- NO --> C12[END]
    end

```

The flowchart illustrates the game logic, starting with character and foe creation, followed by location determination, and then the main game loop. The loop includes checking for character and foe status (HP, MP) and handling various game events based on coordinates.

```

graph TD
    Start([Start]) --> PrintStars[/print stars in the list to help user to make a choice/]
    PrintStars --> AskChoices[/Ask user for a number of choices/]
    AskChoices --> AcceptInput[/accept user input and convert to integer and assign to user_choice/]
    AcceptInput --> IsUserChoiceValid{is user_choice valid?}
    IsUserChoiceValid -- YES --> ReturnValid([RETURN valid])
    IsUserChoiceValid -- NO --> AskValid[/Ask user for a valid number of choices/]
    AskValid --> AcceptValid[/accept user input and convert to integer and assign to user_choice/]
    AcceptValid --> IsUserChoiceValid

    IsUserChoiceValid --> IsNorth{is user choice North and character's x-coordinate is not 0?}
    IsNorth -- YES --> ValidN[valid = True]
    ValidN --> IsNorth
    IsNorth -- NO --> IsEast{is user choice East and character's x-coordinate is not 24?}
    IsEast -- YES --> ValidE[valid = True]
    ValidE --> IsEast
    IsEast -- NO --> IsSouth{is user choice South and character's y-coordinate is not 24?}
    IsSouth -- YES --> ValidS[valid = True]
    ValidS --> IsSouth
    IsSouth -- NO --> IsWest{is user choice West and character's x-coordinate is not 0?}
    IsWest -- YES --> ValidW[valid = True]
    ValidW --> IsWest
    IsWest -- NO --> ReturnValid

    IsNorth --> IsKTrue{is k True?}
    IsKTrue -- YES --> GenRand[/generate random number in [1, 4] and assign to damage_by_foe/]
    GenRand --> CharHP[character's HP = damage_by_foe]
    CharHP --> PrintInflict[/print explanation that how much damage was inflicted/]
    PrintInflict --> DisplayHPN[DISPLAY_CHARACTER_HP(character)]
    DisplayHPN --> End1([END])
    IsKTrue -- NO --> DisplayHPS[DISPLAY_CHARACTER_HP(character)]
    DisplayHPS --> PrintSuccess[/print explanation that character ran away successfully/]
    PrintSuccess --> End1

    IsWest --> IsUserChoiceN{is user choice North?}
    IsUserChoiceN -- YES --> CharX1[character's x-coordinate = 1]
    CharX1 --> IsUserChoiceE{is user choice East?}
    IsUserChoiceE -- YES --> CharX2[character's x-coordinate = 1]
    CharX2 --> IsUserChoiceS{is user choice South?}
    IsUserChoiceS -- YES --> CharY1[character's y-coordinate = 1]
    CharY1 --> IsUserChoiceW{is user choice West?}
    IsUserChoiceW -- YES --> CharX3[character's x-coordinate = 1]
    CharX3 --> End2([END])
    IsUserChoiceW -- NO --> End2
    IsUserChoiceS -- NO --> End2
    IsUserChoiceE -- NO --> End2
    IsUserChoiceN -- NO --> End2
  
```

ACTION - COMBAT

```
graph TD
    COMBAT([COMBAT (character, foe)]) --> Define[define character, die, foe_runaway = False]
    Define --> UserChoice[get user_choice = 0]
    UserChoice --> Decision1{is character not die and foe not die and foe not runaway and user_choice = 0?}
    Decision1 -- YES --> COMBAT_ROUND[COMBAT_ROUND (character, foe)]
    Decision1 -- NO --> IsFoeDead{is foe dead?}
    IsFoeDead -- YES --> PrintSituation[print a situation of the situation that character won the fight]
    PrintSituation --> FoeRunaway[character runs away]
    FoeRunaway --> CheckLevel[CHECK_FOR_LEVEL_UP (character, foe)]
    CheckLevel --> IsFoeRunaway{is foe run away?}
    IsFoeRunaway -- YES --> PrintRunaway[print foe run away message]
    IsFoeRunaway -- NO --> IsCharacterDead{is character dead?}
    IsCharacterDead -- YES --> PrintRunaway
    IsCharacterDead -- NO --> IsUserWantToFightAway{is user want to fight away?}
    IsUserWantToFightAway -- YES --> CharacterFlee[CHARACTER_FLEE (character)]
    CharacterFlee --> END([END])
    IsUserWantToFightAway -- NO --> GetUserChoice[GET_USER_CHOICE (player chooses fight or flee)]
    GetUserChoice --> IsUserWantToFightAway

    COMBAT_ROUND --> GenerateRandomNumber[generate random number in the range 1, 100 and assign to foe_roll]
    GenerateRandomNumber --> IsCharacterRollGreater[is character_roll greater than foe_roll?]
    IsCharacterRollGreater -- YES --> COMBAT_STRIKE[COMBAT_STRIKE (foe, character, foe)]
    IsCharacterRollGreater -- NO --> PrintToss[print "tossing successfully strikes away"]
    PrintToss --> IsFoeHPGreater[is foe's HP > 0?]
    IsFoeHPGreater -- YES --> COMBAT_STRIKE
    IsFoeHPGreater -- NO --> PrintFoeDead[print "Foe is dead"]
    PrintFoeDead --> IsFoeHPLessOrEqualZero[foe's HP <= 0?]
    IsFoeHPLessOrEqualZero -- YES --> FoeDie[foe_die = True]
    FoeDie --> IsCharacterHPLessOrEqualZero[is character's HP <= 0?]
    IsCharacterHPLessOrEqualZero -- YES --> FoeRunaway = True
    FoeRunaway = True --> IsFoeNameBoss[is foe name is Foe_BOSS_NAME? and foe not Foe_AWAY?]
    IsFoeNameBoss -- YES --> FoeRunaway = False
    IsFoeNameBoss -- NO --> FoeRunaway = True
    FoeRunaway = True --> RETURN_Foe_Die([RETURN foe_die, character_die, foe_runaway])
    FoeRunaway = False --> IsCharacterHPLessOrEqualZero
    IsCharacterHPLessOrEqualZero -- YES --> PrintCharacterDead[print "character is dead"]
    PrintCharacterDead --> COMBAT_STRIKE
    IsCharacterHPLessOrEqualZero -- NO --> IsFoeHPGreater

    COMBAT_STRIKE --> HighRoller{is high_roller character, foe}
    HighRoller -- YES --> CharacterAttack[CHARACTER_ATTACK_DESCRIPTION (character, foe)]
    CharacterAttack --> CharacterRollMax[character_roll_max of CHARACTER_DAMAGE_POINTS (character)]
    CharacterRollMax --> GenerateRandomNumber2[generate random number in the range and assign to foe_inflicted_damage]
    GenerateRandomNumber2 --> FoeHP[foe's HP]
    FoeHP --> FoeInflictedDamage[foe_inflicted_damage]
    FoeInflictedDamage --> DisplayFoeHP[DISPLAY_FOE_HP(foe)]
    DisplayFoeHP --> DisplayCharacterHP[DISPLAY_CHARACTER_HP(character)]
    DisplayCharacterHP --> END
    HighRoller -- NO --> FoeAttack[FOE_ATTACK_DESCRIPTION (character, foe)]
    FoeAttack --> GenerateRandomNumber3[generate random number in the range and assign to character_inflicted_damage]
    GenerateRandomNumber3 --> CharacterHP[character's HP]
    CharacterHP --> CharacterInflictedDamage[character_inflicted_damage]
    CharacterInflictedDamage --> PrintCharacterHP[print character's HP]
    PrintCharacterHP --> DisplayCharacterHP
    DisplayCharacterHP --> DisplayFoeHP

    CHARACTER_ATTACK_DESCRIPTION --> IsFoeCharacterClassScore[is foe character class score?]
    IsFoeCharacterClassScore -- YES --> Description1[description = list which consists of key-value pairs of the dictionary 'Successor skill' they should be correspond to character's level]
    Description1 --> IsFoeCharacterClassSkill[is foe character class skill?]
    IsFoeCharacterClassSkill -- YES --> Description2[description = list which consists of key-value pairs of the dictionary 'Thief skill' they should be correspond to character's level]
    Description2 --> IsFoeCharacterClassBowman[is foe character class bowman?]
    IsFoeCharacterClassBowman -- YES --> Description3[description = list which consists of key-value pairs of the dictionary 'Bowman skill' they should be correspond to character's level]
    Description3 --> PrintAttached[print character attached foe with some skill]
    PrintAttached --> END
    IsFoeCharacterClassScore -- NO --> IsFoeCharacterClassSkill
    IsFoeCharacterClassSkill -- NO --> IsFoeCharacterClassBowman
    IsFoeCharacterClassBowman -- NO --> PrintAttached

    CHARACTER_DAMAGE_POINTS --> IsCharacterClassScore[is the character class score?]
    IsCharacterClassScore -- YES --> SetCharacterDPScore[set character's dp as score and dp which is a constant tuple]
    SetCharacterDPScore --> IsCharacterClassSkill[is the character class skill?]
    IsCharacterClassSkill -- YES --> SetCharacterDPSkill[set character's dp as skill and dp which is a constant tuple]
    SetCharacterDPSkill --> IsCharacterClassBowman[is the character class bowman?]
    IsCharacterClassBowman -- YES --> SetCharacterDPBowman[set character's dp as bowman and dp which is a constant tuple]
    SetCharacterDPBowman --> RETURN_CHARACTER_DP([RETURN character's dp])
    IsCharacterClassScore -- NO --> IsCharacterClassSkill
    IsCharacterClassSkill -- NO --> IsCharacterClassBowman
    IsCharacterClassBowman -- NO --> RETURN_CHARACTER_DP

    COMBAT_WITH_FINAL_BOSS --> CreateBoss[CREATE_BOSS (character)]
    CreateBoss --> BossASCII[BOSS_ASCII]
    BossASCII --> GetUserChoice2[GET_USER_CHOICE (player choose fight or flee)]
    GetUserChoice2 --> IsUserWantToFight[is user_choice = fight?]
    IsUserWantToFight -- YES --> CharacterDieBossDie[character_die, boss_die, foe_runaway = False]
    CharacterDieBossDie --> IsBossNotDie[is boss not die and character not die?]
    IsBossNotDie -- YES --> CharacterDieBossDie
    IsBossNotDie -- NO --> IsCharacterHPEqualOrSmaller[is character's hp equal to or smaller than 0?]
    IsCharacterHPEqualOrSmaller -- YES --> RETURN_CHARACTER_DIE([RETURN "character_die"])
    IsCharacterHPEqualOrSmaller -- NO --> PrintByeMessage[print bye message to user]
    PrintByeMessage --> RETURN_GAME_ONGOING([RETURN game_ongoing])
```