Intro:

The player wakes up in a dark room, with a faint glow visible beneath a doorway on the far side of the room. With no recollection of prior events, or their current location, the player must use their observation and quick wits to escape the building they awake in. Sinister things lurk in the darkness, and strange noises break the eerie silence on occasion.

Action Scenarios:

At certain moments in the exploration of their surroundings, or items in the rooms they explore, players will be forced to make a choice from up to 3 different options. These create major or minor branches which play a role in whether the player is able to escape. Choices may impact the options available to players during combat scenarios.

Quest:

The player must escape the building to clear the game. Additional exploration of the building and its rooms/contents is not necessary for completion, but it will provide contextual clues about the situation.

**Pseudo-code**

**General Functions:**

SetPlayerStats – prompts player to set values for character sex, character name, character speed, character luck and character strength. A limited number of points (5) can be allocated between the last 3 categories.

MinimumStat – function enables or disables available choices based on the point allocation of speed, luck, and strength. Returns true or false if the requirements for choices are met by the current player stat setup. Allows game to fail the player for making choices they cannot handle.

TransferX – function to transfer a quantity of X items or currency between the player and merchants, chests or fallen enemies. Before transfers, check that currency is available for the purchase, and that inventory space exits for the items. Also check item count in the stock it is being taken from.

PrintX – Takes in a string to display in the console.

ChoiceFor – uses PrintX to display a prompt, then gets a choice from 1 to 3. The players choice is passed back to the program.

LuckyChoice – uses random number generation to return true if lucky, and false if regular. When true, applies critical damage to attacks. Called on entering a new room, to decide the extent of items the player notices.

PrintStock – displays the contents of the inventory it is called on. (merchant, chests, player, enemy corpses)

PricingX – takes in an item, and a merchant discrimination value. Discrimination value is used to adjust true prices from the itemPrices array to achieve the merchant’s price. 0 discrimination returns true item price.

StateCheck – check the states of object it is called on. Prints current HP, and equipped weapons/armour to console.

**Combat:**

Battle – Void function. Loops as long as the player has not escaped, or

BattleEnd – checks the player stats (health, luck) and stats of all enemies in the encounter to determine the battle’s state or outcome. Battles have the following conclusions:

1. Ongoing – enemies are still in the battle instance, total enemy health is greater than zero, total player health is greater than zero. The attack turn is passed from players to enemies depending on the Speed stat.
2. Ended – total enemy health is zero, or player health reduce to zero. “Battle Complete” or “Game Over” displayed appropriately. Victory in battle shows the inventory list of all enemies, allowing the player to choose their loot.
3. Escaped – the player successfully fled from battle. No loot is presented at the end. Enemy count on that floor remains un-altered. Successful escapes depend on the Luck and Speed stats

TurnOrder – compares the Speed stast of the player and enemies to determine the order of turns in combat. Manages who the next turn goes to.

Attack – function reduces the hp of the targeted object/enemy. Damage is calculated based on player stats. Strength increases damage, luck increases the chance for a critical strike, speed increases the chance of receiving no damage from an attack (evasion chance).

Escape – causes the player/enemy it is called on to attempt escaping the battle. Successful attempts depend on RNG and Luck stat.

EnemyAction – controls how an enemy behaves during its turn. Enemies with key items may try to flee depending on how player’s average damage per turn. Enemies may choose to attack, guard, charge up a more powerful attack for their next turn, or flee.

IsCritical – uses the LuckyChoice function to determine whether an attack is a critical hit. Called in the Attack() function before damage is applied.

EvadeCheck – uses the speed stat of the attacker and the luck stat of player/enemy under attack. Returns true or false. If true, changes damage of an attack to 0.

**Classes:**

**Being**:

Class contains properties: health, speed, luck, strength. All stored as integers.

Contains Attack(), Block(), Escape(), and all character actions.

Contains functions to Set attributes

Contains functions that return attributes

**Stocked:**

Adds Vector<string> to hold inventory, and an integer to hold currency.

Contains attribute ‘costChange’, set to 1 by default. (Merchants can increase costs).

Contains TransferX function, PrintX function.

**Game Management:**

Bool roomPlan {false, false, false)

Initialized with false at the start of each room, for the number of choices available. Used to prevent repeat choices.

Int roomProgress = 3 (value not final)

Decrements as player choices are made. Once value reaches 0, end room battle/boss, or progression to next room occurs. This is reset at the beginning of each room.  
Players are aware of how many choices they may make before progressing to the level end.

Int gameProgress = 10; (value not final)

This integer is decremented for each room progressed. A separate case statement group will control the activities in each room.

Case GameState:

1. GetChoice – Statements in this case prompt the player for a choice, using appropriate string inputs
2. Battle – Starts battles, Passes the battle turn amongst beings in the current battle
3. Merchant – This case manages trade and treasure collection
4. Room Progress – controls the progression in the game. Calls functions to display text.

Case Room:

Cases here contain the events of each room, such as items presented to the player, battles that will occur, treasure that can be found.

Sample room events:

1. Room entry. Present overview of room contents, such as item on the left, enemy on the right, ladder straight ahead. (Displayed as text).
2. Prompts for user choice. Decrements the roomProgress count. The bool for that choice is set to true if it is false. If it was already true, display error message, and redo the choice.
3. Uses Case statements to run appropriate choice. All case statements are within a do-while loop, that breaks when the roomProgress count reaches 0
4. After roomProgress do-while loop room end or room battles remaining take place.