1. **Data Structures**
   1. Format of I/O files:
      1. Player saves:
         1. Encrypted with DES
         2. Files consist of information about the save (timeline, resources, inventory, player stats, etc..)
      2. Game mechanic files
         1. Story files consisting of plot messages, events prompts, monster descriptions, etc… in different files for each type, plot messages sorted in order chronological order
         2. Resource files consist of stats for items, including weapons, resources, buildings, etc... sorted by type
   2. Classes:
      1. GameRunner
         1. main(): initializes GUI, Game class, etc..
      2. Game:
         1. runGame(): runs all necessary methods and initializes all values to begin game
            1. Loads file; if file not found, starts from very beginning
            2. Initializes all values
         2. saveFile(): saves user data to file
            1. Runs string through encrypt method of cipher class
            2. Saves string to file
         3. loadFile(): loads user data from file
            1. Loads encrypted string from file
            2. Runs string through decrypt method of cipher class
            3. Initializes all values, as determined through decrypted string
      3. Cipher: see recursion below for pseudocode
         1. encrypt(int): returns encrypted user data using DES
         2. encrypt(boolean): returns encrypted user data using DES
         3. decryptToInt(String): returns decripted user data using DES
         4. decryptToBoolean(String): returns decrypted user data using DES
      4. A Dusky Path
         1. Item[ ] storage: used to store inventory of user
         2. Events events: used to keep track of events encountered by the user
         3. explore(): allows player to “explore”, which allows the player to encounter the events of the game
         4. loot(): allows player to “loot” enemies
            1. Utilizes Enemy giveLoot() method and Player receiveLoot(Item) method
      5. The Room
         1. boolean[ ] buildingsUnlocked: keeps track of the unlocked buildings
         2. Item[ ] resources: keeps track of resources collected by player
         3. buildBuilding(): allows player to construct a building
         4. buildArmor(): allows player to construct armor
         5. buildWeapons(): allows player to construct weapons
      6. The Village
         1. Workers[ ] jobs: keeps track of the workers of the village
         2. changeJob(int, int): used to assign a worker a job
         3. gatherResources(): gathers resources
      7. *Entity*
         1. int hpMax: maximum hp that the entity can have
         2. int hpLeft: current hp that the entity has
         3. double hitChance: the probability that the entity will do damage
         4. receiveDamage(int): takes in an integer value and decreases hpLeft by value
         5. attack(Entity): runs attack against explicit entity
            1. Player

Item[ ] store:

attack(Entity): performs an attack on an enemy

heal(): heals player

prepare(): prepares player to enter ADuskyPath, and allows player to select equipment, etc..

sortInventory(): sorts player inventory by type

receiveLoot(Loot): uses separateItems() method in Loot and adds Item[ ] to store

* + - * 1. Enemy

int damage: amount of damage enemy does

Loot loot: loot that enemy drops upon death

giveLoot(Entity): returns an instance of Loot

attack(Entity): returns an integer value of damage

* + 1. Events
       1. int encounters: number of encounters player has
       2. generateEncounter(): creates a fight using randomized method to  
           generate Enemy object
       3. generateLoot(): returns loot when called
    2. Fight
       1. Player player: main player in fight
       2. Enemy enemy: enemy in fight
       3. battle(): runs combat between two entities
       4. consumeMeat(): heals player during combat
    3. Loot
       1. Item[ ] items: array of items that loot consists of
       2. separateItems(): returns Item[ ]
    4. Item
       1. int amount: the number of these items
       2. boolean unlocked: whether the item is unlocked by player or not
       3. compareToItem(Item): returns integer value of comparison between two items. Used to sort inventory
       4. deduct(int): deducts number from amount
       5. retrieve(int): returns an item with an amount, and deducts that  
           amount from this item
          1. Uses deduct(int) method
       6. add(int): adds number to amount
    5. Resources
       1. String name: name of the resource
       2. compareToAmount(Resources): returns integer value of comparison between two resources
    6. Weapon
       1. int strength: the strength of the weapon
       2. compareTo(Weapon): returns integer value of comparison between two weapon strengths
    7. Healing
       1. int healed: amount healed for
    8. *Workers:*
       1. *gatherResources(): adds resources to inventory*
          1. Gatherer

final Resources PLUSWOOD

* + - * 1. Hunter

final Resources PLUSMEAT

final Resources PLUSHIDE

* + - * 1. Steelworker

final Resources PLUSSTEEL

final Resources MINUSCOOKEDMEAT

* + - * 1. Chef

final Resources PLUSCOOKEDMEAT

final Resources MINUSMEAT

* + - * 1. Trapper

final Resources PLUSBAIT

final Resources MINUSMEAT

* + - * 1. Ironworker

final Resources PLUSIRON

final Resources MINUSCOOKEDMEAT

1. **Algorithms:**
   1. Searching and Sorting:
      1. Bubble sort used to sort player inventory

From the first item in inventory to the second last

Compare from the current to the last to see if the item

current item ranks before or after the next item

Move to appropriate position if necessary

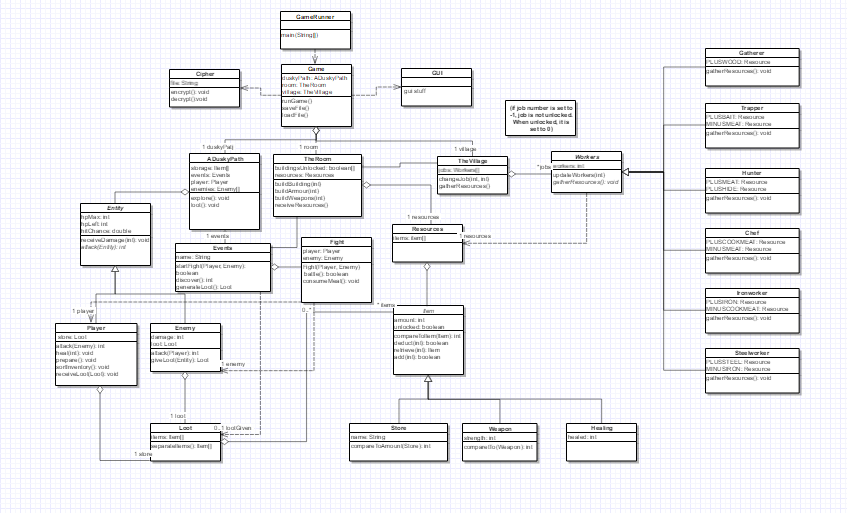
Exit if no more changes

* + 1. Linear search used to find data from file, or find resource from inventory

From first item to last item  
 If item == search item  
 Return item

* 1. Recursion:
     1. Encryption for integers
        1. Break integer into blocks of set length, record number of blocks recreated
           1. Ex. for blocks of 3: 23456 to 23 and 456, record 2
        2. Perform set mathematical operation of additions and multiplications to each of the data blocks
           1. Ex. add 3, times 5, add 7, times 2 to 23 and then to 456
        3. Repeat steps 1 and 2 for each of the new numbers generated for set amount of steps
        4. Record the final set of integers as String
     2. Decryption for integers
        1. Receive recorded number of blocks of numbers generated for each data
        2. Decrypt the following chunks of numbers with 1 less repetition in the decryption
        3. For each repetition
           1. Perform the reverse mathematical operation performed in the encryption process
           2. Combine the resulting numbers
           3. Repeat until the correct number of decryption process is finished
        4. Return final integer result
     3. Encryption for booleans
        1. Receive a boolean to be encypted
        2. Using a set of boolean operations, generate a random set of booleans that after evaluation, returns the orignal boolean
           1. Ex. for the operation boolean && boolean || boolean, an input of T can be encrypted randomly into TTF, TTT, TFT, FTT, FFT such that each of the encrypted boolean sets will evaluate into T
        3. Repeat the process for each of the new booleans in the generated set for a set amount of times
        4. Record the series of booleans as a String
     4. Decryption for booleans
        1. Receive a String of booleans
        2. Evaluate the String
           1. Using recursion, evaluate the string by breaking the string down into segments of equal length such that the total number of segments is equal to the number of booleans generated from one encryption process and evaluating each segment
           2. If the number of booleans is equal to the number of booleans generated , evaluate the expression using the original formula
        3. Return the final boolean result
  2. Battle
  3. Other:

1. **UML**



For clearer image: (<https://www.gliffy.com/go/share/sl6d09prsvo56ulkhjij>)

1. **PseudoCode:**

(Assuming accessors and mutators added where necessary)

**TheVillage**

* gatherResources()  
   for each item i on the list

Use gatherResources for each worker on the list

* changeJobs(int: index, int: number)

If there are workers available

Use the update workers method for base workers gatherer and the worker class you are updating

**Workers**

* gatherResources()

(the following is the same for all subclasses of workers)

Search resources for item to increase

Increase resource by amount indicated by constant

Search resources for item to decrease

Decrease resource by amount indicated by constant

* updateWorkers(int): a mutator method

**TheRoom**

* buildBuilding(int)

if resources variable is greater than or equal to amount needed for building

buildingsUnlocked[int] set to true

* buildArmor(int)

if resources variable is greater than or equal to amount needed for armor

Increase player health by corresponding amount

* buildWeapons(int)

If resources variable is greater than or equal to amount needed for weapon

Add instance of weapon to player’s store[ ] item array

* receiveResources()

Use gatherResources() method from TheVillage/Workers

**Events**

* startFight(Player, Enemy):boolean

Create a Fight object using Player and Enemy

Call the battle method from Fight

Return the boolean generated from the battle method

* generateLoot(): Loot

Read preset chances for items from a file

Randomize an integer num

Create an Item[] of size num

For each Item i in the list

Randomize Item chosen and number given

Instantiate a loot object with the item[ ]

Return the loot

* discover(): int

Read preset chances for a location discovery from a file

Generate a random integer

Utilize integer to choose location to discover

Return index of unlocked building to be utilized by TheRoom

**Entity**

* recieveDamage(int): void

Takes in an integer and reduces the implicit Entity’s health by that integer.

**Player**

* attack(Enemy): void

Takes in an entity which the implicit player “attacks”

Finds the “weapon (item)” in the player’s store

Calls recieveDamage(int) using the enemy parameter and int from weapon

* heal(int): void

Takes in a int which “heals” the implicit player by that integer

* prepare(): void

Lets the player choose items from their store (inventory)

1 Weapon, 1 Armour, 1 Type of healing(as many as they want)

* Resources (up to max storage amount which is determined by a storage item)

**Enemy**

* attack(Player): int

Take in player which the entity “attacks”

Deals given amount of damage

Calls receiveDamage of player

* giveLoot(Entity): Loot

Drops loot store from predefined values

**Item**

* compareToItem(Item): int  
   Return difference between amount in implicit item and explicit item
* deduct(int): boolean  
   Deducts integer value from amount

Returns false if value is initially 0

* retrieve(int): Item  
   if int greater than or equal to amount  
   deduct(int)  
   return new Item with amount = int
* add(int): boolean  
   if int + amount < max storage  
   Adds int to amount  
   Return true  
   Else  
   Return false

**Store**

compareTo: item

Returns the item with the most quantity

**Weapons**

compareToDamage(Weapon): int

Returns the difference between the weapon damage

**Healing**

**Loot**

* separateItems(): items[]

Sort Method to sort items(probably insertion)

**Fight**

* battle(): boolean

While player is alive and enemy is alive

If player choose to attack

Call attack method of player

Else if player choose to heal

If the player chooses cured meat

Call consumeMeat method

Call attack method of enemy

Return if the enemy dies

* consumeMeat(): void

Try to deduct 1 cured meat from inventory

If successful, call heal method of Player using the healed field of the cured

meat class

**Cipher**

* encrypt()

If item encrypted is an integer

Break integer into blocks of set length, record number of blocks recreated

* + - 1. Ex. for blocks of 3: 23456 to 23 and 456, record 2

Perform set mathematical operation of additions and multiplications to each of the data blocks

* + - 1. Ex. add 3, times 5, add 7, times 2 to 23 and then to 456

Repeat steps 1 and 2 for each of the new numbers generated for set amount of steps

Record the final set of integers as String

If item encrypted is a boolean

Receive a boolean to be encrypted

Using a set of boolean operations, generate a random set of booleans that after evaluation, returns the original boolean

* + - 1. Ex. for the operation boolean && boolean || boolean, an input of T can be encrypted randomly into TTF, TTT, TFT, FTT, FFT such that each of the encrypted boolean sets will evaluate into T

Repeat the process for each of the new booleans in the generated set for a set amount of times

Record the series of booleans as a String

* decrypt()

If item being decrypted is an integer

Receive recorded number of blocks of numbers generated for each data

Decrypt the following chunks of numbers with 1 less repetition in the decryption

For each repetition

* + - 1. Perform the reverse mathematical operation performed in the encryption process
      2. Combine the resulting numbers
      3. Repeat until the correct number of decryption process is finished

Return final integer result

If item being decrypted is a boolean

Receive a String of booleans

Evaluate the String

* + - 1. Using recursion, evaluate the string by breaking the string down into segments of equal length such that the total number of segments is equal to the number of booleans generated from one encryption process and evaluating each segment
      2. If the number of booleans is equal to the number of booleans generated , evaluate the expression using the original formula

Return the final boolean result