1. Theme/Quest

1.1 Design:

The main Quest of the game will be implemented using a series of Kismet triggers which will check to see if the player has gathered all the required artifacts in order to open the final door, by using this trigger we allow the player to complete the dungeons in the order they desire. Kill events, triggering locations and picking up items will also be tracked using the quest manager, and event manager.

1.2 Traceability:

Systems are implemented, content varies. Kismet code controlling button sequence for door opening, as well as key collection for door opening. UnrealScript: TS_QuestManager, TS_QuestMenu, TS_HUD_Quests, TS_SubQuest, TS_EventManager.

2. World

2.1 Design:

The map layouts of our game will be created in the UDK, the items and props which populate the world will be mainly static meshes which will be created in Blender. Kismet will be used to create triggers within the world such as spawning enemies and interacting with other NPCs.

2.2 Traceability:

Systems are implemented, content varies. UnrealScript: TS_ConversationManager, TS_Pawn. Kismet code controlling the actor factories for the level and trigger geometry to call traps and spawns.

3. Story

3.1 Design:

Story is currently handled by conversations with NPCs, carried out through the use of UnrealScript. The first part of our story, the “missing people story-arc” will be implemented using Kismet scripts which will allow the player to communicate with NPCs to solve the mystery of the missing people. These scripts simplify the implementation of dialog trees which means we can create dialog trees with more layers or paths. The second part of our story, the world lore, will consist of Kismet triggers to spawn certain enemies in specific locations, allowing the player to explore the map and discover what is hidden inside our world. Some of the triggers will allow the player to discover some information which will help find the reason for the missing people.

3.2 Traceability:

Systems are implemented, content varies. UnrealScript: TS_ConversationManager, TS_QuestManager, TS_EventManager. Kismet trigger geometry.

4. Gameplay

4.1 Design:

Each level is handled by a unique UDK map, with common assets shared between all maps. Players are able to pick up items from enemies and use them to augment their own abilities. The gameplay is centered around solving puzzles as well as avoiding traps or destroying enemies. The levels will be designed to try and surprise the player with a different challenge, be it different types of enemies or a new type of puzzle, to make the player consider many possible strategies. As explained in the Core Mechanics section, the player can choose to assign tablets to different stats, meaning he can change from a lot of Health to a lot of Damage with the click of a button, which is why we will have puzzles and traps which will push the player to use the ability to change attributes quickly. The player can become affected by various spells, spell effects, passive effects and stat bonuses, uniquely affecting gameplay.

4.2 Traceability:

UDK maps TS-Temple-Matt.udk and TS-Traps.udk contain different level layouts and trap concepts. UnrealScripts: BonusInfo, PassiveSpellInfo, SpellInfo, StatusEffectInfo, TS_Curse, TS_Fireball, TS_Heal, TS_InventoryMenu, TS_Items, TS_Lightning, TS_Lightning_Horizon, TS_Poisonball, TS_Projectile, TS_StatusEffect_Burning, TS_StatusEffect_Holy, TS_StatusEffect_Electric, TS_StatusEffect_Poisoned, TS_StatusEffect_Zombie, TS_StatusEffects, TS_Trap, TS_Trap_Darts, TS_Trap_Spear, TS_FallingBlock.

5. Gameplay Modes

5.1 Design:

The modes of gameplay (dungeon play and exploration) are handled in slightly different ways. In dungeon play, the player is able to attack most NPCs in the various dungeons built with the UDK level designer. The dungeons will have no more than 3 possible branching paths per path to reduce design complexity for both the player (reduce navigation confusion) and for the designers (making some changes to the level will not require too much overhead). Furthermore, the number of traps per level (medium sized) will be within the realm of 10-15, enough that the levels are challenging but manageable for the player to remember upon playthroughs. The traps themselves will be handled using kismet event triggers. The combat actions are initiated using the mouse controls (left click to attack). However, in exploration mode, conversations are initiated with friendly NPCs by being within a specific radial distance and using an interact action (bound to a specific keyboard stroke). The player will also be unable to attack the villagers while in towns, preventing players from possibly halting progress in the game. Additionally, the player

will be able to interact with lore objects by being within a certain radial distance from an object and using the same interact action. These interactions will be determined by specific .uc files holding scripted dialogue and AI behaviour for each individual friendly NPC and a general AI behaviour for each grouping of enemy NPC. When engaging with friendly NPCs, the dialog boxes that appear will be rendered using scaleform UI tools, allowing the player to choose decisions through the use of Actionscript 2 events and relaying the decisions to interact with specific .uc files. The player will generally not have more than 3-4 choices when conversing with an NPC, this is to simplify the overall complexity of managing conversation trees for both the player (too many choices is a bad thing) and the designers (too many branches makes one NPC conversation very hard to change). Regarding the inventory system, the interface will be designed using scaleform (flash), Actionscript 2 and a relay to a .uc file about inventory management, similar to that of the NPC dialog. The inventory system will initially have a limit on the number of tablets you can hold at once, but the number of tablets you can equip is going to be fixed around 1-5 for the player. The purpose of limiting the capacity of tablets that a player can hold is to add a level of depth to the item management system/skill system, if a player can hold everything at all times, they make no sacrifices. This prevents the game from becoming too easy with no consequences for gameplay choices. The tablets that an enemy drops will be controlled in the .uc file of the specific type of enemy, based on some type of RNG, and a specific percent chance for each type of tablet. This will give the game an element of replayability and add an optional amount of grinding for specific types of tablets, prolonging the time spent in game, and providing the player with another motive to kill enemies.

5.2 Traceability:

UnrealScript: TS_Trap, TS_Trap_Darts, TS_Trap_Spear, TS_ConversationManager, TSHUD, GFxHUD, TS_InventoryMenu, TS_Items. Kismet trigger geometry.

6. Core Mechanics

6.1 Design:

The core mechanics of the game are those of a 3rd person top down isometric hack and slash. The user controls the player character using the keyboard for movement (WASD or arrow keys) and using the mouse buttons for attacking. The main attributes of the character are health, attack damage, attack speed, and defense. These attributes can be set in the .uc file of the main character. In addition, enemy units will also have these attributes set in their respective .uc files. The modification of these attributes through picking up tablets will be done by the game modifying the variables pertaining to the attributes using a separate .uc file which inherits from the original attributes file. The levelling up of certain items will be done by using an item class which all individual items inherit from, simply, there will be a variable that denotes what level the current item is at, and a multiplier of that level with certain attributes of the item. These Unreal Script files are all implemented using UDK game types.

6.2 Traceability:

UnrealScript: TS_Pawn, MyPawn, MyPlayerController, BonusInfo, SpellInfo, PassiveSpellInfo,

StatusEffectInfo, TS_Weapon, TSWeap_Sword, TSWeap_Sword_Ammo, TSWeap_Sword_Attach, TSWeap_Sword_Controller, TS_ChaosItem, TS_CurseItem, TS_DamageItem, TS_FireballItem, TS_HealItem, TS_HealthItem, TS_HealthRegenItem, TS_InventoryMenu, TS_StatMenu, TS_Items, TS_LightningItem, TS_ManaItem, TS_ManaRegenItem, TS_MultishotItem, TS_PoisonballItem, TS_SpeedItem, TS_StormItem, TS_WorldItemPickup.

7. Characters

7.1 Design:

Characters in game are each extended from the same base class, with enemies, NPCs, and the player each having control of their own groups of pawns to utilize. These classes are written entirely in UnrealScript and called from Kismet and the UDK map editor. They use assets defined for them in the UPK_Characters package, which includes models, textures, physics and animation assets. All characters in game will consist of a skeletal mesh and appropriate models, the skeletal meshes will be used by many characters and the materials will be changed in order to minimize the size of the game file. These characters will fit into one of two sets: humans and monsters. Humans (including the player character) will make use of variations of the same model and textures, while monster characters use models and textures that are easily distinguishable as separate from the from the humans. A member of any particular group cannot hurt other members of the same group, but are free to damage any members of the opposite group. That is, humans can't hurt each other but are able to hurt monsters, and vice versa. The purpose of human characters are to assist the player (who is also a human) by advancing the plot and augmenting the player's abilities, while the purpose of monster characters are to act as an obstacle to challenge the player as they progress through the game.

7.2 Traceability:

UnrealScripts: TS_Pawn, MyPawn, MyPlayerController, TS_Enemy_Ratman, TS_Enemy_Ratman_Controller

8. Presentation

8.1 Design:

The game's default camera angle will be defined by Kismet. It will always be looking at the player model and will be fixed to the player model's relative position, with a small offset to the X and Y coordinates and a large one to the Z coordinate. This will create a top-down camera that will look down on the player with a pitch angle between 45° and 90°. Level geometry will be designed to accommodate the camera's position above the player, such that unimportant details do not obstruct the player's camera and important objects are not hidden from view. A scaleform HUD will be applied on top of the screen, and a mouse cursor will be visible to show its position on the screen relative to the player.

8.2 Traceability:

UnrealScripts: TS_Pawn, MyPawn, TSHUD, GFxHUD, TS_NPCWorldDisplay, TS_PauseMenu, TS_QuestMenu, TS_StatMenu, TS_InventoryMenu, TS_HUD_Quests.

9. Levels

9.1 Design:

There will be five maps - One for the NPC village, one for the overworld, one for the tutorial dungeon and one for the first dungeon, in addition to a fifth which will save as the interiors of any buildings that the player wishes to enter. The player will start the game in the NPC village and find themselves forced into the tutorial dungeon. From then on, the restrictions on the player are lifted and they are freely able to travel between any of the four maps. Players
will move from area to area by walking through a doorway which triggers a kismet sequence that will load the corresponding map.

9.2 Traceability:

UDK maps TS-Temple-Matt.udk and TS-Traps.udk are the only map files that currently fit the design, and both of them are designed to contain dungeon elements.