Temp notes for wiki

Just gonna use this page for organizing some new content – it will eventually go into other pages of this wiki.

- Mudbill

Engine scripts

A list of script functions available in A Machine For Pigs (that are not present in The Dark Descent)

General

```
void CheckPoint(string@& asName, string@& asStartArea, string@& asFunction,
string@& asTextCat, string@& asTextEntry, const bool abPlayerLimbo)
```

Creates a check point where the player will respawn if they die. **Callback syntax:** `void MyFunc(string &in asName, int alCount)`
This function replaces TDD's `CheckPoint`.

1. *asName* - The internal name of the callback
2. *asStartArea* - The name of the PlayerStartArea to respawn at
3. *asFunction* - The name of the callback function to run
4. *asTextCat* - The name of the .lang file category (SEEMS UNUSED)
5. *asTextEntry* - The name of the .lang file entry (SEEMS UNUSED)
6. *abPlayerLimbo* - Whether to put the player into a state of “limbo” upon death. In limbo, the death occurs normally, but once the screen fades to black, it remains so until `ReleasePlayerFromLimbo` is called. Player is still controllable during death.

```
void ReleasePlayerFromLimbo();
```

Respawns the player if stuck in “limbo”. Limbo can be enabled with `CheckPoint`. Limbo is the state between life and death. The player enters limbo upon dying but won't respawn until this function is called.

```
void PlayScriptedAnimation(string@& asEntity, string@& asAnimation, const
bool abLoop);
```

Plays an animation that has been added to an entity through the Model Editor. Similar to `PlayEnemyAnimation`.

1. asEntity - Internal name of the entity
2. asAnimation - The animation name inside the entity
3. abLoop - Whether to loop the animation or play it only once

Screen effects

```
void ShowScreenImage(string@& asImageFile, int alX, int alY, const float afUnknown1, const bool abUnknown2, const float afDuration, const float afFadeInTime, const float afFadeOutTime);
```

Shows a 2D image on the screen. Originally used for showing the intro logo during a sequence.

1. *asImageFile* - The image file to show. Does not have to be pow2 (although it gives a warning if not)
2. *alX* - The X position of the top left corner, starting from the center of the screen. Use negative half of the width of the image to center it.
3. *alY* - The Y position of the top left corner, starting from the center of the screen.
4. *afUnknown1* - Unsure what this is. Any value 0 or greater results in the image not showing up, so use -1
5. *abUnknown2* - Unsure what this is. If set to true, image does not display, so use false
6. *afDuration* - How long, in seconds, the image displays. This does not include fade times
7. *afFadeInTime* - How long, in seconds, the fade in spends
8. *afFadeOutTime* - How long, in seconds, the fade out spends

Journal

```
void AddHint(string& asNameAndTextEntry, string@& asImage);
```

Adds a hint to the player's journal.
This function replaces TDD's `AddDiary`.

1. *asNameAndTextEntry* - The .lang text entry of the hint. The entry must be in category “Journal” and begin with “Hintckgedit> and end with _ckgedit_QUOTName” for the title and end with “_Text” for the body.
2. *asImage* - This argument does not seem to be used. Presumed to be for displaying an image but possibly removed functionality. All existing occurrences are empty.

```
void SetJournalDisabled(const bool abDisabled);
```

Disables the player's ability to open their journal.

1. *abDisabled* - True to disable, false to enable again

Player

```
void SetLanternFlickerActive(const bool abActive);
```

Enables/disables the flicker effect for the lantern's light.

1. *abActive* - True to flicker, false to not flicker

```
void SetPlayerInfection(const float afInfectionLevel);
```

Sets the infection level for the player. Infection replaces sanity from TDD and acts similar, however it goes from 0-100 instead of 100-0. An infection level above approximately 20 will affect the player's ability to move. Infection level above 80 will kill the player. Infection slowly decreases over time, unless high enough.
This function replaces TDD's `SetPlayerSanity`.

1. *afInfectionLevel* - The level of infection to set

```
void FadePlayerPitchTo(const float afPitch, const float afDeaccelleration,
const float afSpeed);
```

Moves the player's pitch (up and down rotation).

1. *afPitch* - The target pitch to move towards. 0 = straight forward. Clamped to range: -70 to +70.
2. *afDeaccelleration* - The deaccelleration when nearing the target pitch. A low value makes a slow change.
3. *afSpeed* - The speed of the movement. Speed is affected by deaccelleration.

```
void SetPlayerUsesDragFootsteps(const bool abX);
```

Sets whether the player's footstep sounds are replaced with “drag” versions. Some surface materials do not have drag steps, and will therefore play no footstep sounds.

1. abX - Whether to use “drag” step sounds

Entities

```
void SetPhysicsAutoDisable(string@& asEntity, const bool abDisabled);
```

Does... something. Not immediately apparent. Is only used on chandeliers in the campaign.

1. *asEntity* - The entity to affect
2. *abDisabled* - Whether this effect auto disables or not

```
void SetLampFlickerActive(string@& asLamp, const bool abActive);
```

Enables a flickering effect on a Lamp-type entity.

1. *asLamp* - The lamp entity
2. *abActive* - Whether to enable flicker

```
void StartPhoneRinging(string@& asEntity);
```

Enables a PhoneBox-type entity to start ringing. A ringing phone box can be interacted with to play some audio files. After interacting, the phone will stop ringing.

1. *asEntity* - The PhoneBox entity

```
void StopPhoneRinging(string@& asEntity)
```

Stops a ringing PhoneBox-type entity.

1. *asEntity* - The PhoneBox entity

```
bool GetEntityActive(string@& asEntity);
```

Returns whether an entity in the level is active or not.

1. *asEntity* - The entity to check

```
void StopPropAnimation(string@& asProp);
```

Stops the animation currently playing on a prop. Animations can be started with `PlayPropAnimation`.

1. *asProp* - The name of the entity/prop

```
void SetPropAnimationPosition(string@& asProp, const float afPosition);
```

Jumps to a specifc point in an animation. Generally used in conjunction with `PlayPropAnimation`.

1. *asProp* - The entity that is being animated
2. *afPosition* - The time within the animation, in seconds, to jump to

```
void SetSwingDoorOpenAmount(string@& asEntity, const float afOpenAmount,
const float afTime, const bool abUnknown);
```

Sets the open amount for a swing door.

1. *asEntity* - The SwingDoor entity
2. *afOpenAmount* - The new amount state to set. Range: 0 - 1
3. *afTime* - The time in seconds until the door has changed state
4. *abUnknown* - Unsure what this does. If set to true, nothing happens (?), so use false.

```
void FadeLampTo(string@& asEntity, const uint alR, const uint alG, const
uint alB, const uint alA, int alRadius, const double afTime);
```

Fades a Lamp-type entity's light to another color. This function uses integers for color values instead of floats, which is a little odd. Likewise, it uses a double floating point for the last argument instead of a regular one.

1. *asEntity* - The lamp to change
2. *alR* - Red value (appropriate values are 0 - 10)
3. *alG* - Green value (appropriate values are 0 - 10)
4. *alB* - Blue value (appropriate values are 0 - 10)
5. *alA* - Alpha value (has no effect?)
6. *alRadius* - The new radius to use (affects illumination strength)
7. *afTime* - Time in seconds until the light properties have changed

```
void SetButtonCanBeSwitchedOn(string@& asEntity, const bool abX);
```

Changes whether a Button-type entity can be toggled.

1. *asEntity* - The button entity

2. *abX* - Whether it can be switched on

```
void CreateEntityAtArea(string@& asName, string@& asFile, string@& asArea,
const bool abUnknown, const float afOffsetX, const float afOffsetY, const
float afOffsetZ, const float afRotX, const float afRotY, const float
afRotZ);
```

Creates an entity at an area with the specified offsets and rotation.
This function replaces TDD's `CreateEntityAtArea`.

1. *asName* - The internal name of the created entity
2. *asFile* - The file for the entity (extension .ent)
3. *asArea* - The area to create entity at
4. *abUnknown* - Not sure. Only true is used in the campaign. Seemingly has no effect
5. *afOffsetX* - The offset along the X axis in units
6. *afOffsetY* - The offset along the Y axis in units
7. *afOffsetZ* - The offset along the Z axis in units
8. *afRotX* - The rotation on the X axis in degrees
9. *afRotY* - The rotation on the Y axis in degrees
10. *afRotZ* - The rotation on the Z axis in degrees

```
void AttachPropToBone(string@& asProp, string@& asEntity, string@& asBone,
const float afOffsetX, const float afOffsetY, const float afOffsetZ, const
float afRotX, const float afRotY, const float afRotZ);
```

Attaches a prop to a specific bone within another entity. You can inspect bones in the Model Editor.
Note: Offsets and rotations are local to the bone and relative to its rotation.

1. *asProp* - The prop to attach
2. *asEntity* - The entity that holds the bone to attach to
3. *asBone* - The bone within the entity to attach to
4. *afOffsetX* - Offset along the X axis
5. *afOffsetY* - Offset along the Y axis
6. *afOffsetZ* - Offset along the Z axis
7. *afRotX* - Rotation along the X axis
8. *afRotY* - Rotation along the Y axis
9. *afRotZ* - Rotation along the Z axis

```
void DetachPropFromBone(string@& asProp);
```

Detaches an attached prop. Note: When detached, physics are not automatically enabled on the prop.

1. *asProp* - The attached prop

Sounds

```
void AddEffectVoice2(string@& asVoiceFile, string@& asEffectFile, string@&
asTextCat, string@& asTextEntry1, const float afStartTime1, string@&
asTextEntry2, const float afStartTime2, const bool abUsePosition, string@&
asPosEntity, const float afMinDistance, const float afMaxDistance);
```

Plays an audio file with 2 consecutive subtitles.

1. *asVoiceFile* - The entire voice file to play (intended to include 2 sections)
2. *asEffectFile* - The background effect file to play during voices. Leave empty for no extra effect
3. *asTextCat* - The .lang text category for the subtitles
4. *asTextEntry1* - The first .lang subtitle entry
5. *afStartTime1* - The time to wait until the first subtitle starts
6. *asTextEntry2* - The second .lang subtitle entry
7. *afStartTime2* - The time to wait until the second subtitle starts
8. *abUsePosition* - Whether to use 3D to play the sound from an entity
9. *asPosEntity* - The entity to play the sound from. If empty, plays from player
10. *afMinDistance* - The minimum distance required between the player and the entity in order to hear the audio
11. *afMaxDistance* - The maximum distance allowed between the player and the entity in order to hear the audio

```
void AddEffectVoice3(string@& asVoiceFile, string@& asEffectFile, string@& asTextCat, string@& asTextEntry1, const float afStartTime1, string@& asTextEntry2, const float afStartTime2, string@& asTextEntry3, const float afStartTime3, const bool abUsePosition, string@& asPosEntity, const float afMinDistance, const float afMaxDistance);
```

Plays an audio file with 3 consecutive subtitles.

1. *asVoiceFile* - The entire voice file to play (intended to include 3 sections)
2. *asEffectFile* - The background effect file to play during voices. Leave empty for no extra effect
3. *asTextCat* - The .lang text category for the subtitles
4. *asTextEntry1* - The first .lang subtitle entry
5. *afStartTime1* - The time to wait until the first subtitle starts
6. *asTextEntry2* - The second .lang subtitle entry
7. *afStartTime2* - The time to wait until the second subtitle starts
8. *asTextEntry3* - The third .lang subtitle entry
9. *afStartTime3* - The time to wait until the third subtitle starts
10. *abUsePosition* - Whether to use 3D to play the sound from an entity
11. *asPosEntity* - The entity to play the sound from. If empty, plays from player
12. *afMinDistance* - The minimum distance required between the player and the entity in order to hear the audio
13. *afMaxDistance* - The maximum distance allowed between the player and the entity in order to hear the audio

```
void AddEffectVoice4(string@& asVoiceFile, string@& asEffectFile, string@& asTextCat, string@& asTextEntry1, const float afStartTime1, string@& asTextEntry2, const float afStartTime2, string@& asTextEntry3, const float afStartTime3, string@& asTextEntry4, const float afStartTime4, const bool abUsePosition, string@& asPosEntity, const float afMinDistance, const float afMaxDistance);
```

Plays an audio file with 4 consecutive subtitles.

1. *asVoiceFile* - The entire voice file to play (intended to include 4 sections)
2. *asEffectFile* - The background effect file to play during voices. Leave empty for no extra effect

3. *asTextCat* - The .lang text category for the subtitles
4. *asTextEntry1* - The first .lang subtitle entry
5. *afStartTime1* - The time to wait until the first subtitle starts
6. *asTextEntry2* - The second .lang subtitle entry
7. *afStartTime2* - The time to wait until the second subtitle starts
8. *asTextEntry3* - The third .lang subtitle entry
9. *afStartTime3* - The time to wait until the third subtitle starts
10. *asTextEntry4* - The fourth .lang subtitle entry
11. *afStartTime4* - The time to wait until the fourth subtitle starts
12. *abUsePosition* - Whether to use 3D to play the sound from an entity
13. *asPosEntity* - The entity to play the sound from. If empty, plays from player
14. *afMinDistance* - The minimum distance required between the player and the entity in order to hear the audio
15. *afMaxDistance* - The maximum distance allowed between the player and the entity in order to hear the audio

```
void AddEffectVoice5(string@& asVoiceFile, string@& asEffectFile, string@&
asTextCat, string@& asTextEntry1, const float afStartTime1, string@&
asTextEntry2, const float afStartTime2, string@& asTextEntry3, const float
afStartTime3, string@& asTextEntry4, const float afStartTime4, string@&
asTextEntry5, const float afStartTime5, const bool abUsePosition, string@&
asPosEntity, const float afMinDistance, const float afMaxDistance);
```

Plays an audio file with 5 consecutive subtitles.

1. *asVoiceFile* - The entire voice file to play (intended to include 5 sections)
2. *asEffectFile* - The background effect file to play during voices. Leave empty for no extra effect
3. *asTextCat* - The .lang text category for the subtitles
4. *asTextEntry1* - The first .lang subtitle entry
5. *afStartTime1* - The time to wait until the first subtitle starts
6. *asTextEntry2* - The second .lang subtitle entry
7. *afStartTime2* - The time to wait until the second subtitle starts
8. *asTextEntry3* - The third .lang subtitle entry
9. *afStartTime3* - The time to wait until the third subtitle starts
10. *asTextEntry4* - The fourth .lang subtitle entry
11. *afStartTime4* - The time to wait until the fourth subtitle starts
12. *asTextEntry5* - The fifth .lang subtitle entry
13. *afStartTime5* - The time to wait until the fifth subtitle starts
14. *abUsePosition* - Whether to use 3D to play the sound from an entity
15. *asPosEntity* - The entity to play the sound from. If empty, plays from player
16. *afMinDistance* - The minimum distance required between the player and the entity in order to hear the audio
17. *afMaxDistance* - The maximum distance allowed between the player and the entity in order to hear the audio

```
void AddEffectVoice6(string@& asVoiceFile, string@& asEffectFile, string@&
asTextCat, string@& asTextEntry1, const float afStartTime1, string@&
asTextEntry2, const float afStartTime2, string@& asTextEntry3, const float
afStartTime3, string@& asTextEntry4, const float afStartTime4, string@&
asTextEntry5, const float afStartTime5, string@& asTextEntry6, const float
afStartTime6, const bool abUsePosition, string@& asPosEntity, const float
```

```
afMinDistance, const float afMaxDistance);
```

Plays an audio file with 6 consecutive subtitles.

1. *asVoiceFile* - The entire voice file to play (intended to include 6 sections)
2. *asEffectFile* - The background effect file to play during voices. Leave empty for no extra effect
3. *asTextCat* - The .lang text category for the subtitles
4. *asTextEntry1* - The first .lang subtitle entry
5. *afStartTime1* - The time to wait until the first subtitle starts
6. *asTextEntry2* - The second .lang subtitle entry
7. *afStartTime2* - The time to wait until the second subtitle starts
8. *asTextEntry3* - The third .lang subtitle entry
9. *afStartTime3* - The time to wait until the third subtitle starts
10. *asTextEntry4* - The fourth .lang subtitle entry
11. *afStartTime4* - The time to wait until the fourth subtitle starts
12. *asTextEntry5* - The fifth .lang subtitle entry
13. *afStartTime5* - The time to wait until the fifth subtitle starts
14. *asTextEntry6* - The sixth .lang subtitle entry
15. *afStartTime6* - The time to wait until the sixth subtitle starts
16. *abUsePosition* - Whether to use 3D to play the sound from an entity
17. *asPosEntity* - The entity to play the sound from. If empty, plays from player
18. *afMinDistance* - The minimum distance required between the player and the entity in order to hear the audio
19. *afMaxDistance* - The maximum distance allowed between the player and the entity in order to hear the audio

```
void AddEffectVoice7(string@& asVoiceFile, string@& asEffectFile, string@&
asTextCat, string@& asTextEntry1, const float afStartTime1, string@&
asTextEntry2, const float afStartTime2, string@& asTextEntry3, const float
afStartTime3, string@& asTextEntry4, const float afStartTime4, string@&
asTextEntry5, const float afStartTime5, string@& asTextEntry6, const float
afStartTime6, string@& asTextEntry7, const float afStartTime7, const bool
abUsePosition, string@& asPosEntity, const float afMinDistance, const float
afMaxDistance);
```

Plays an audio file with 7 consecutive subtitles.

1. *asVoiceFile* - The entire voice file to play (intended to include 7 sections)
2. *asEffectFile* - The background effect file to play during voices. Leave empty for no extra effect
3. *asTextCat* - The .lang text category for the subtitles
4. *asTextEntry1* - The first .lang subtitle entry
5. *afStartTime1* - The time to wait until the first subtitle starts
6. *asTextEntry2* - The second .lang subtitle entry
7. *afStartTime2* - The time to wait until the second subtitle starts
8. *asTextEntry3* - The third .lang subtitle entry
9. *afStartTime3* - The time to wait until the third subtitle starts
10. *asTextEntry4* - The fourth .lang subtitle entry
11. *afStartTime4* - The time to wait until the fourth subtitle starts
12. *asTextEntry5* - The fifth .lang subtitle entry
13. *afStartTime5* - The time to wait until the fifth subtitle starts
14. *asTextEntry6* - The sixth .lang subtitle entry

15. *afStartTime6* - The time to wait until the sixth subtitle starts
16. *asTextEntry7* - The seventh .lang subtitle entry
17. *afStartTime7* - The time to wait until the seventh subtitle starts
18. *abUsePosition* - Whether to use 3D to play the sound from an entity
19. *asPosEntity* - The entity to play the sound from. If empty, plays from player
20. *afMinDistance* - The minimum distance required between the player and the entity in order to hear the audio
21. *afMaxDistance* - The maximum distance allowed between the player and the entity in order to hear the audio

Enemies

```
void AddEnemyPatrolNode(string@& asEnemy, string@& asPathNode, const float afWaitTime, string@& asAnimation, const bool abUnknown)
```

Adds a patrol node to the enemy's walking path. A path is restarted from the beginning when the final node is reached. Note: Inputting an invalid animation in asAnimation at the final node will make the enemy wait there indefinitely.
This function replaces TDD's AddEnemyPatrolNode.

1. *asEnemy* - The name of the enemy
2. *asPathNode* - Internal name of path node
3. *afWaitTime* - The time, in seconds, the enemy waits at this node before continuing. Note: A time of 0.0f does not seem to skip waiting, use 0.01f instead if you want the enemy to immediately continue to the next node.
4. *asAnimation* - The animation to play on the enemy when they arrive at this path node. Animations can be found in the Model Editor. Leave empty to play no special animation (uses default Idle animation). Note: If the animation lasts longer than afWaitTime, the enemy waits until the animation is complete before continuing the path.
5. *abUnknown* - Unknown variable. Only false is ever used in the campaign. If set to true, seems to affect how animations are played, however they seem to just stutter or loop.

```
void SetEnemyMoveType(string@& asEnemy, string@& asMoveType);
```

Changes how an enemy moves.

1. *asEnemy* - The name of the enemy
2. *asMoveType* - The type to change to. Type can be “WalkBiped”, “RunBiped”, “ChargeBiped”, “Idle”

```
void SetManPigType(string@& asEnemy, string@& asType);
```

Sets the type for a ManPig enemy. It is unknown whether this function does anything or if it's just left over from an earlier state of the game. Only “Freddy” is used as the type, but supposedly it should also accept “Rod” and “Jane”.

1. *asEnemy* - The ManPig enemy.
2. *asType* - The type to set. Type can be “Freddy”, “Rod”, “Jane”

```
void PlayEnemyAnimation(string@& asEnemy, string@& asAnimation, const bool
```

```
abLoop, const float afDelay);
```

Plays a specific animation for an enemy.

1. *asEnemy* - Internal name of the enemy (asterisk is allowed)
2. *asAnimation* - The name of an animation registered to the enemy
3. *abLoop* - Whether the animation loops
4. *afDelay* - Seems to affect how the animation plays out. A higher value makes the animation slower, although it seems to also skip some keyframes or perhaps merge them, making the animation look incorrect. Experiment to see what works based on the animation.

```
void ChangeEnemyPose(string@& asEnemy, string@& asPose);
```

Changes the pose for an enemy. Can be either “Biped” or “Quadruped”.

1. *asEnemy* - Internal name of the enemy
2. *asPose* - The pose to change to

```
void ForceEnemyWaitState(string@& asEnemy);
```

Forces the enemy's AI to change the state to “Wait” which makes the enemy wait for a short while before continuing its' normal actions. An enemy without patrol nodes defaults to the “Wait” state. Otherwise, if patrol nodes are added, the enemy will continue the path after waiting is done.

1. *asEnemy* - Internal name of the enemy

```
void SetEnemyBlind(string@& asEnemy, const bool abX);
```

Sets whether the enemy can see the player if they are within visible range.

1. *asEnemy* - Internal name of the enemy
2. *abX* - Whether enemy is blind

```
void SetEnemyDeaf(string@& asEnemy, const bool abX);
```

Sets whether the enemy can hear the player make sound if they are within audible range.

1. *asEnemy* - Internal name of the enemy
2. *abX* - Whether enemy is deaf

```
bool GetPlayerCanSeeEnemy(string@& asEnemy);
```

Returns whether the enemy is within visible range of the player.

1. *asEnemy* - Internal name of the enemy

```
float GetEnemyPlayerDistance(string@& asEnemy);
```

Returns the distance (in HPL units) between the enemy and the player.

1. asEnemy - Internal name of the enemy

Particles

```
void SetParticleSystemActive(string@& asParticleSystem, const bool
abActive);
```

Pauses a particle system in its current frame. The paused particle system remains frozen at this frame until reactivated or destroyed.

1. *asParticleSystem* - The name of the particle system
2. *abActive* - False to pause, true to unpause

```
void DestroyParticleSystemInstantly(string@& asParticleSystem);
```

Destroys a particle system and any existing particles already emitted from it. Similar to `DestroyParticleSystem`, except that one will not destroy the existing particles and rather let them live out their lives. This function will cut all particles' lives short.

1. *asParticleSystem* - The PS to destroy