GPP_Plugin Tech Info [v4]

IExamInterface Functions

- **WorldInfo World_GetInfo() const;**
  //Returns a *WorldInfo* structure which contains world information (see Exam_HelperStructs.h)

- **StatisticsInfo World_GetStats() const;**
  //Returns a StatisticsInfo structure which contains several useful game statistics (see Exam_HelperStructs.h)

- **bool Fov_GetHouseByIndex(UINT index, HouseInfo& houseInfo) const;**
  //Used to retrieve the HouseInfo from houses inside the FOV (see Exam_HelperStructs.h)
  //Return TRUE: HouseInfo found for given index
  //Return FALSE: No HouseInfo found for given index (= no more houses after this index)
  → Use Plugin::GetHousesInFOV() which uses this function, but returns a vector of HouseInfos

- **bool Fov_GetEntityByIndex(UINT index, EntityInfo& enemyInfo) const;**
  //Used to retrieve the EntityInfo from entities (items & enemies) inside the FOV (see Exam_HelperStructs.h)
  //Return TRUE: EntityInfo found for given index
  //Return FALSE: No EntityInfo found for given index (= no more entities after this index)
  → Use Plugin::GetEntitiesInFOV() which uses this function, but returns a vector of EntityInfos

- **AgentInfo Agent_GetInfo() const;**
  //Returns a AgentInfo structure which is packed with crucial agent parameters (see Exam_HelperStructs.h)

- **bool Enemy_GetInfo(EntityInfo entity, EnemyInfo& enemy);**
  //Used to retrieve some additional information about an enemy (Type, Health, …)
  //Returns TRUE: Enemy with given entityHash found. EnemyInfo is set.
  //Returns FALSE: No enemy found. Note that enemies receive new hashes between frames, using the same entityHash the next frame won't work.

- **Vector2 NavMesh_GetClosestPathPoint(Vector2 goal) const;**
  //Returns the next point of the path towards the given goal, using the Navigation Mesh

- **bool Item_GetInfo(EntityInfo entity, ItemInfo& item);**
  //Use this function to inspect an item (EntityInfo → ItemInfo). You can use this ItemInfo to request more information about the item using the Weapon_GetAmmo, Medkit_GetHealth and FOood_GetEnergy function
  //The function uses the EntityHash to lookup the corresponding Item

- **bool Item_Grab(EntityInfo entity, ItemInfo& item);**
  //Use this function to grab an item/entity (EntityInfo → ItemInfo)
  //The function uses the EntityHash to lookup the corresponding Item, items must be inspected and added to the inventory during the same frame.
  //The EntityInfo argument is ignored when DebugParams.**AutoGrabClosestItem is active**

- **bool Item_Destroy(EntityInfo entity);**
  //Use this function to destroy an entity (only items)
  //The function uses the EntityHash to lookup the corresponding Item, item must be in FOV and grabrange of the agent

- **int Weapon_GetAmmo(ItemInfo& item);**
  //Use this function to request the amount of ammo an item of type [PISTOL] has.
  //This function can be used with items from the inventory or items still on the ground (not yet picked up)
  //Returns -1 if invalid item is used

- **int Medkit_GetHealth(ItemInfo& item);**
  //Use this function to request the amount of health an item of type [MEDKIT] has.
  //This function can be used with items from the inventory or items still on the ground (not yet picked up)
  //Returns -1 if invalid item is used

- **int Food_GetEnergy(ItemInfo& item);**
  //Use this function to request the amount of energy an item of type [FOOD] has.
  //This function can be used with items from the inventory or items still on the ground (not yet picked up)
  //Returns -1 if invalid item is used

- **bool PurgeZone_GetInfo(EntityInfo entity, PurgeZoneInfo& zone);**
  //Used to retrieve some additional information about a purge zone (Center, Radius)
  //Returns TRUE: PurgeZone with given entityHash found. PurgeZoneInfo is set.
  //Returns FALSE: No PurgeZone found.

- **bool Inventory_AddItem(UINT slotId, ItemInfo item);**
  //This function adds the given item to the inventory @ the given slotid
  //Given slot must be empty. ItemInfos are acquired through Item_Grab(...)

- **bool Inventory_RemoveItem(UINT slotId);**
  //Removes the item at the given inventory slot

- **bool Inventory_GetItem(UINT slotId, ItemInfo& item);**
  //Retrieves the ItemInfo from the item stored at the given inventory slot

- **bool Inventory_UseItem(UINT slotId);**
  //Use the item at the given inventory slot. Some items can be used, others can't (Check console output)
  //Returns false when use fails (because of reasons mentioned in the console) OR when the item ran out of charges (no ammo left)

- **UINT Inventory_GetCapacity() const;**
  //Returns the inventory capacity (amount of inventory slots)

- **Vector2 Debug_ConvertScreenToWorld(Vector2 screenPos) const;**
  //Converts a screenposition to worldposition using the camera's VP

- Vector2 Debug_ConvertWorldToScreen(Vector2 worldPos) const;
  //Converts a worldposition to screenposition using the camera's VP

- bool Input_IsKeyboardKeyDown(InputScancode key) const;
  //Returns if the specified keyboard key was pressed down

- bool Input_IsKeyboardKeyUp(InputScancode key) const;
  //Returns if the specified keyboard key was released

- bool Input_IsMouseButtonDown(InputMouseButton button) const;
  //Returns if the specified mouse button was pressed down

- bool Input_IsMouseButtonUp(InputMouseButton button) const;
  //Returns if the specified mouse button was released

- MouseData Input_GetMouseData(InputType type, InputMouseButton button) const;
  //Returns the mouse data for the specified mouse button and type.

- void Draw_...
  //Set of Draw commands which can be for debugging/visualization

GameDebugParams

Several properties can be controlled or overridden by changing the GameDebugParams (see Exam_HelperStructs.h). These are set during the Plugin::InitGameDebugParams call inside the Plugin class. NOTE that these parameters are only used during a debug build, default values are used when running the release version. So these parameters can only be used for debugging purposes, and your final AI should be test with the default values (or in a release build).

```
bool SpawnEnemies = true; //Spawn enemies?
int EnemyCount = 20; //Amount of enemies?
int ItemCount = 40; //Amount of items?
bool GodMode = false; //Use GodMode? (Invincible)
bool IgnoreEnergy = false; //Ignore energy depletion
bool AutoFollowCam = false; //Auto follow the player
bool RenderUI = false; //Render Player UI (Parameters)
bool AutoGrabClosestItem = false; //Auto Grab closest item (Item_Grab)
string LevelFile = "LevelOne.gppl"; //Level to load?
int Seed = 1234; //Seed for random generator
int StartingDifficultyStage = 0; // Overwrites the difficulty stage
bool InfiniteStamina = false; // Agent has infinite stamina
bool SpawnDebugPistol = false; // Spawns pistol with 1000 ammo at start
bool SpawnPurgeZonesOnMiddleClick = false; // Middle mouse clicks spawn purge zones
```

Include Files

The project contains several include files, located in the 'inc' folder. These files are 'shared includes' which are also used to build the Host-Programs & BasePlugin Library. Altering these include files will (in most cases, depending on what you change and where it is used) result in **heavy crashes** and/or **undefined behavior**. It's important that the structures that are shared between the plugin and host-program are identical. Changing the structures means changing them only for the plugin, and not for the host-program.

Files that can't be changed (none of the include files actually):

- inc/Exam_HelperStructs.h
- inc/IBaseInterface.h
- inc/IExamInterface.h
- inc/IExamPlugin.h
- inc/IPluginBase.h

If you want to extend a structure, create a sub structure that inherits from the original structure. example:

```
struct SteeringPlugin_Output
{
    Elite::Vector2 LinearVelocity = { 0.f,0.f };
    float AngularVelocity = 0.f;
    bool AutoOrientate = true;
    bool RunMode = false;
};
```

```
struct SteeringPlugin_Output_Extended: SteeringPlugin_Output
{
    bool IsSet = false;
};
```