GOAP in TOMB RAIDER

Chris Conway
Lead AI Engineer, Crystal Dynamics

GOAP in Tomb Raider

- Started in 2006 for unannounced title at the request of our lead designer, based on his impressions from the GOAP presentation at GDC 2006.
  - First shipped title using it was Tomb Raider 2013 (and now being used for the next Tomb Raider due in 2015).

GOAP in Tomb Raider

- Started in 2006 for unannounced title at the request of our lead designer, based on his impressions from the GOAP presentation at GDC 2006.
  - First shipped title using it was Tomb Raider 2013 (and now being used for the next Tomb Raider due in 2015).

- Most AI development time is spent on code support for Goals and Actions, not the GOAP library code.

GOAP in Tomb Raider

- Started in 2006 for unannounced title at the request of our lead designer, based on his impressions from the GOAP presentation at GDC 2006.
  - First shipped title using it was Tomb Raider 2013 (and now being used for the next Tomb Raider due in 2015).
- Most AI development time is spent on code support for Goals and Actions, not the GOAP code.
- New GOAP features added as gameplay requirements changed…

Extensions to GOAP

- Situational Costs for Actions
  - Causes a Plan with an inexpensive primary Action to potentially become expensive due to high situational cost of an Action for a requirement.

Extensions to GOAP

- Situational Costs for Actions
  - Causes a Plan with an inexpensive primary Action to potentially become expensive due to high situational cost of an Action for a requirement.
  - For example:
    - MeleeAttack(cost=1) + Goto(cost=20, due to 20m of movement) = Plan Cost of **21**
    - RangedAttack(cost=10) + Goto(cost=1, due to 1m of movement) = Plan Cost of **11**

Extensions to GOAP

- Situational Costs for Actions
  - Causes a Plan with an inexpensive primary Action to potentially become expensive due to high situational cost of an Action for a requirement.
  - For example:
    - MeleeAttack(cost=1) + Goto(cost=20, due to 20m of movement) = Plan Cost of **21**
    - RangedAttack(cost=10) + Goto(cost=1, due to 1m of movement) = Plan Cost of **11**
    - Preferable to putting hard limits on attacks because there might be situations where some attacks are unavailable (e.g. out of ammo, melee weapon destroyed, etc.).

Situational Costs for Actions, continued…

- Can also be used to vary the costs of Actions depending on equipment or some other modifier.
  - RangedAttack(cost=5 with MachineGun, or 10 with a Bow).
  - MeleeAttack(cost=1 with Dual-wielded Swords, or 5 with Fists).

• Situational Costs for Actions, continued…

- Can also be used to vary the costs of Actions depending on equipment or some other modifier.
  - RangedAttack(cost=5 with MachineGun, or 10 with a Bow).
  - MeleeAttack(cost=1 with Dual-wielded Swords, or 5 with Fists).

- Useful for creating competing complex (multi-Action) methods of solving the same requirement(s).
  - Goto(cost=20, for 20m) + MeleeAttack(cost=1) = Plan Cost of **21**
  - TakeOff(cost=1) + FlyTo(cost=4, for 20m) + Land(cost=1) + MeleeAttack(cost=1) = Plan Cost of **7**

Motives to drive Goal/Action Availability and Cost

- Very useful for an Action like UseObject that must determine which, of many, objects an NPC should consider using.
  - The object can "advertise" an effect that is different from the change that will happen after actual usage.
  - The system can be tuned to modify motives over time (e.g. Hunger always increases on its own), or in response to events, in addition to before/during/after Actions such as UseObject.

Motives to drive Goal/Action Availability and Cost

- Very useful for an Action like UseObject that must determine which, of many, objects an NPC should consider using.
  - The object can "advertise" an effect that is different from the change that will happen after actual usage.
  - The system can be tuned to modify motives over time (e.g. Hunger always increases on its own), or in response to events, in addition to before/during/after Actions such as UseObject.

- Motives can also be used to control Goals.
  - For Example, the Investigate Goal might only be available if a motive named Suspicion is higher than some tunable value.

Motives to drive Goal/Action Availability and Cost

- Very useful for an Action like UseObject that must determine which, of many, objects an NPC should consider using.
  - The object can “advertise” an effect that is different from the change that will happen after actual usage.
  - The system can be tuned to modify motives over time (e.g. Hunger always increases on its own), or in response to events, in addition to before/during/after Actions such as UseObject.

- Motives can also be used to control Goals.
  - For Example, the Investigate Goal might only be available if a motive named Suspicion is higher than some tunable value.

- Actions without requirements related to motives can still have an effect on them.
  - For example, a successful attack might reduce Fear, which is used to control the Flee Goal.

Submitting Multiple Plan Candidates

- Optionally, a Goal or Action, such as UseObject, can submit multiple requirement sets (with different values, such as a different object to be used), which the Planner will then add to the options pool (the set of search options the A* search is working on) to find the best plan.

Submitting Multiple Plan Candidates

- Optionally, a Goal or Action, such as UseObject, can submit multiple requirement sets (with different values, such as a different object to be used), which the Planner will then add to the options pool (the set of search options the A* search is working on) to find the best plan.
  - For example, two or three objects might be found for the UseObject Goal that will help address a motive that requires attention, and so we will submit a Requirements List for each, which will then be evaluated as separate plans.
    - The closest object might not necessarily be the best to use, because its requirements might require a more complex plan.

Submitting Multiple Plan Candidates

- Optionally, a Goal or Action, such as UseObject, can submit multiple requirement sets (with different values, such as a different object to be used), which the Planner will then add to the options pool (the set of search options the A* search is working on) to find the best plan.
  - For example, two or three objects might be found for the UseObject Goal that will help address a motive that requires attention, and so we will submit a Requirements List for each, which will then be evaluated as separate plans.
    - The closest object might not necessarily be the best to use, because its requirements might require a more complex plan.
  - Can also be used for multiple targets, to find which one can be attacked for the lowest cost, with an optional (tunable by Goal) plan score multiplier per requirement list to raise the plan score for plans whose requirements list specifies a non-preferential target.

Submitting Multiple Plan Candidates

- Optionally, a Goal or Action, such as UseObject, can submit multiple requirement sets (with different values, such as a different object to be used), which the Planner will then add to the options pool (the set of search options the A* search is working on) to find the best plan.
  - For example, two or three objects might be found for the UseObject Goal that will help address a motive that requires attention, and so we will submit a Requirements List for each, which will then be evaluated as separate plans.
    - The closest object might not necessarily be the best to use, because its requirements might require a more complex plan.
  - Can also be used for multiple targets, to find which one can be attacked for the lowest cost, with an optional (tunable by Goal) plan score multiplier per requirement list to raise the plan score for plans whose requirements list specifies a non-preferential target.
  - Or, for a given Action, two (or more) requirement lists could be submitted (e.g. for AttackRanged we could submit one option to use the currently-equipped bow and another that requires equipping a machine gun first).

Situational Requirements

- Some objects can introduce requirements that must be satisfied by the Planner (in addition to the requirements of the Action that allows usage of that object).

Situational Requirements

- Some objects can introduce requirements that must be satisfied by the Planner (in addition to the requirements of the Action that allows usage of that object).
- For example, a DinnerTable object for the UseObject Goal might have requirements like Food and Drink that can only be acquired by using other objects.
  - A plan to use DinnerTable could look something like this: GoTo(FoodServer), Use(FoodServer) to get Food, GoTo(Bar), Use(Bar) to get Drink, GoTo(DinnerTable), Use(DinnerTable).
  - Performing an Action might affect our inventory and/or state as well (e.g. after using the DinnerTable object the Food and Drink are removed, and the Hunger motive is reduced.

Situational Requirements

- Some objects can introduce requirements that must be satisfied by the Planner (in addition to the requirements of the Action that allows usage of that object).
- For example, a DinnerTable object for the UseObject Goal might have requirements like Food and Drink that can only be acquired by using other objects.
    - A plan to use DinnerTable could look something like this: GoTo(FoodServer), Use(FoodServer) to get Food, GoTo(Bar), Use(Bar) to get Drink, GoTo(DinnerTable), Use(DinnerTable).
    - Performing an Action might affect our inventory and/or state as well (e.g. after using the DinnerTable object the Food and Drink are removed, and the Hunger motive is reduced.
    - Or, for example, the RangedAttack Action might require a certain weapon to be used, and that weapon object will need to be acquired by using another object that advertises that it provides that weapon object, resulting in a Plan containing a UseObject even though RangedAttack doesn't explicitly require it.

Monitoring Child Actions

- Enables a parent Action or Goal to mark a child Action's requirement as completed before the child Action completes on its own.
  - For example, the RangedAttack Action might require that the client must be within 5m of a target, and then monitor the status of the target to force the child Goto Action to terminate early (e.g. if the RangedAttack Action determines that the NPC has LineOfSight to the target within a tunable max ranged attack distance of 10m while the child Action is running).

Monitoring Child Actions

- Enables a parent Action or Goal to mark a child Action's requirement as completed before the child Action completes on its own.
  - For example, the RangedAttack Action might require that the client must be within 5m of a target, and then monitor the status of the target to force the child Goto Action to terminate early (e.g. if the RangedAttack Action determines that the NPC has LineOfSight to the target within a tunable max ranged attack distance of 10m while the child Action is running).

- Also enables the parent Action or Goal to dynamically change the requirements while the child action is in progress.
  - For example, when the target moves it can change the required end position for the child Action, or tell it to move to a new CoverPoint that just became available or viable.
  - This often forces Actions to monitor their requirements dynamically to make sure they are still achievable (and/or requires parent Actions or Goals to make sure they don't modify existing requirements in such a way that completion is no longer possible).

Evaluating and Communicating the Plan’s State

- A Goal or Action can monitor the status of the Plan while child Actions are still running and cause it to abort.
    - For example, a parent Action could determine the target is no longer valid (e.g. he is now injured or dead, or the object we intend to use is no longer available), or it is no longer the best target (e.g. we are now aware of a better target for this Action, so cancel the plan or switch targets, if applicable).

Evaluating and Communicating the Plan's State

- A Goal or Action can monitor the status of the Plan while child Actions are still running and cause it to abort.
    - For example, a parent Action could determine the target is no longer valid (e.g. he is now injured or dead, or the object we intend to use is no longer available), or it is no longer the best target (e.g. we are now aware of a better target for this Action, so cancel the plan or switch targets, if applicable).
- The Goal or Action can also use its awareness of the current state of the Plan to communicate with other NPCs (e.g. notify them that "I'm on my way to that CoverPoint to attack target X" at an appropriate time while the Plan is active).

Evaluating and Communicating the Plan's State

- A Goal or Action can monitor the status of the Plan while child Actions are still running and cause it to abort.
  - For example, a parent Action could determine the target is no longer a valid (e.g. he is now injured or dead, or the object we intend to use is no longer available), or it is no longer the best target (e.g. we are now aware of a better target for this Action, so cancel the plan or switch targets, if applicable).

- The Goal or Action can also use its awareness of the current state of the Plan to communicate with other NPCs (e.g. notify them that "I'm on my way to that CoverPoint to attack target X" at an appropriate time while the Plan is active).

- The Goal or Action can also use this awareness to provide feedback for the player (e.g. "You're in trouble now, I'm going to hit you with a grenade from that hill over there!").

Open-Ended Actions

- Actions don’t necessarily need to complete as soon as they have completed one iteration of the required Action.
  - For example, the RangedAttack Action might be tuned to be allowed to take several shots, and even contain several tasks (such as Reload, StepOutFromCover, StepIntoCover, Aim, Fire, etc.) that it completes while attempting multiple shots.

Open-Ended Actions

- Actions don't necessarily need to complete as soon as they have completed one iteration of the required Action.
    - For example, the RangedAttack Action might be tuned to be allowed to take several shots, and even contain several tasks (such as Reload, StepOutFromCover, StepIntoCover, Aim, Fire, etc.) that it completes while attempting multiple shots.

- Requires accurate maintenance of "remaining cost" for each Action so re-planning can still happen for the same Goal.

Open-Ended Actions

- Actions don't necessarily need to complete as soon as they have completed one iteration of the required Action.
  - For example, the RangedAttack Action might be tuned to be allowed to take several shots, and even contain several tasks (such as Reload, StepOutFromCover, StepIntoCover, Aim, Fire, etc.) that it completes while attempting multiple shots.
- Requires accurate maintenance of "remaining cost" for each Action so re-planning can still happen for the same Goal.
- Prevents repeatedly constructing/starting/finishing an identical Plan for the same Goal, yet we can still find a different Plan with a lower total cost than the remaining cost of the current Plan for that Goal.
  - For example, an NPC is in a great location to shoot from can keep shooting without re-planning for each shot. But, if we find ourselves in a situation where we can make a new Plan for the Attack Goal, such as MeleeAttack, with a lower total cost than the remaining cost of the Plan we already have for that Goal, we are able to do it.

Learning/Adapting based on Success Rates

- The GOAP system in Tomb Raider keeps track of success rates for Goals and Actions, and can be tuned to use that to influence planning.
    - For example, an Action such as MeleeAttack might be tuned to have a variable cost depending on its success rate, which might result in it being used less often (only when the conditions allow the total cost of a Plan with MeleeAttack to be lower than any competing Plan, such as one with RangedAttack, for a given Goal), or only when some other less-expensive option is unavailable (on cooldown, out of ammo, weapon broken, etc.).

Learning/Adapting based on Success Rates

- The GOAP system in Tomb Raider keeps track of success rates for Goals and Actions, and can be tuned to use that to influence planning.
    - For example, an Action such as MeleeAttack might be tuned to have a variable cost depending on its success rate, which might result in it being used less often (only when the conditions allow the total cost of a Plan with MeleeAttack to be lower than any competing Plan, such as one with RangedAttack, for a given Goal), or only when some other less-expensive option is unavailable (on cooldown, out of ammo, weapon broken, etc.).
    - Statistics can be saved with the game data, or reset at tunable intervals or specific milestones. They are tracked as part of the "GOAP Settings" (a set of Goals and Actions with costs and other settings) which are associated with a particular type of NPC.

Learning/Adapting based on Success Rates

- The GOAP system in Tomb Raider keeps track of success rates for Goals and Actions, and can be tuned to use that to influence planning.
  - For example, an Action such as MeleeAttack might be tuned to have a variable cost depending on its success rate, which might result in it being used less often (only when the conditions allow the total cost of a Plan with MeleeAttack to be lower than any competing Plan, such as one with RangedAttack, for a given Goal), or only when some other less-expensive option is unavailable (on cooldown, out of ammo, weapon broken, etc.).
  - Statistics can be saved with the game data, or reset at tunable intervals or specific milestones. They are tracked as part of the "GOAP Settings" (a set of Goals and Actions with costs and other settings) which are associated with a particular type of NPC.
  - Requires different GOAP Settings for different NPC types (so melee-centric NPCs are influenced only by success rates for attacks by other melee-centric NPCs, for example).

Behavior Graph

- We can use a designer-authored Behavior Graph to drive Goal selection, rather than a prioritized list of Goals. This enables designers to create tree- or DAG-like logic for Goal selection depending on the current status of an NPC.

Behavior Graph

- We can use a designer-authored Behavior Graph to drive Goal selection, rather than a prioritized list of Goals. This enables designers to create tree- or DAG-like logic for Goal selection depending on the current status of an NPC.
  - For example, an NPC that is in a pre-combat state might have one set of Goals it considers (with additional logic to cause certain Goals to be evaluated only in certain situations), and another set of Goals for higher alert levels, including highly-customized logic for Goal evaluation in combat situations.

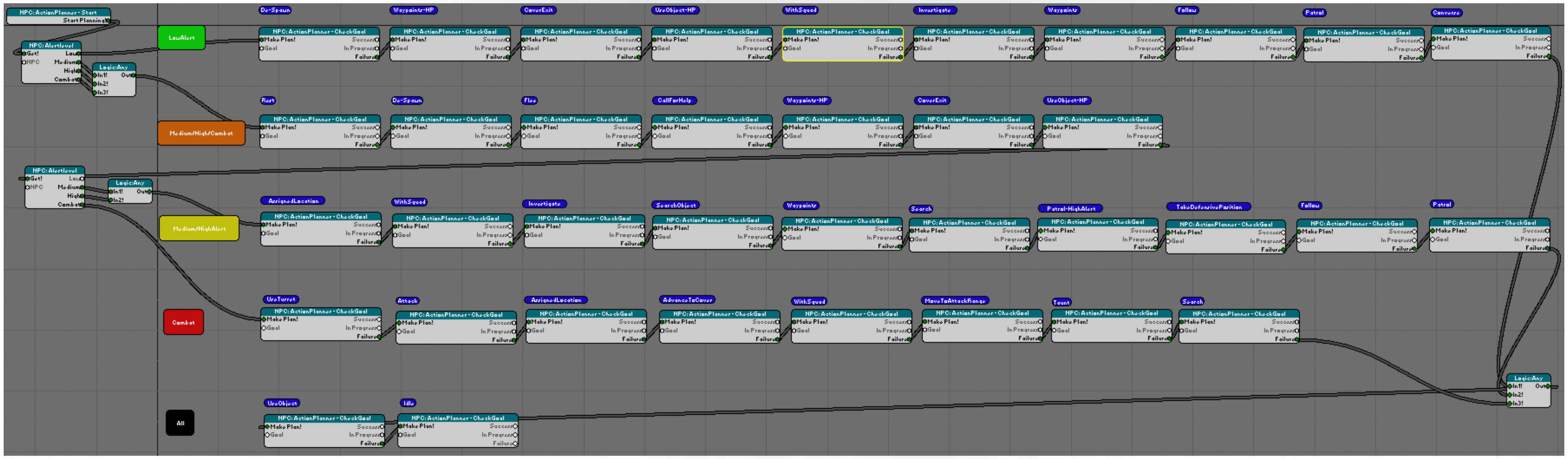
Behavior Graph

- We can use a designer-authored Behavior Graph to drive Goal selection, rather than a prioritized list of Goals. This enables designers to create tree- or DAG-like logic for Goal selection depending on the current status of an NPC.
  - For example, an NPC that is in a pre-combat state might have one set of Goals it considers (with additional logic to cause certain Goals to be evaluated only in certain situations), and another set of Goals for higher alert levels, including highly-customized logic for Goal evaluation in combat situations.
  - Designers can customize usage of certain Goals in the logic of the Behavior Graph without requiring additional code (e.g. Converse will only be available if one or more motives are above or below certain levels), or if the Player has or hasn't done something (ever, or recently), or if the target has a certain weapon equipped, or is in a certain health state, etc.

GDC 15

GAME DEVELOPERS CONFERENCE
MOSCONE CENTER · SAN FRANCISCO, CA
MARCH 2-6, 2015 · EXPO: MARCH 4-6, 2015

Example of a (simplified) Tomb Raider Behavior Graph



TOMB RAIDER

Comprehensive Debugging Support is Critical

- Designers tend to force behaviors (via scripting) when they don’t understand why an NPC is doing something other than what they expected him to do.

Comprehensive Debugging Support is Critical

- Designers tend to force behaviors (via scripting) when they don’t understand why an NPC is doing something other than what they expected him to do.

- Scripting is great for demos or extremely linear sections of the game, but situations without systemic behavior are fundamentally weaker and less interesting, and offer very little replay value.

Debugging Support

- Designers tend to force behaviors (via scripting) when they don't understand why an NPC is doing something other than what they expected him to do.
- Scripting is great for demos or extremely linear sections of the game, but situations without systemic behavior are fundamentally weaker and less interesting, and offer very little replay value.
- Complex plans can be confusing to observe, although complex plans should be the exception, not a common occurrence. This is a fundamental aspect of GOAP: the Planner will always choose the least expensive Plan for a Goal.

Common Question 1: What is that NPC doing right now?

- Give details about the current Plan.
  - When did each Action start?
  - Why did completed Actions finish?
  - Which Action is currently active?
  - What is the status of each Action in the current Plan (including those that haven’t started yet).

Common Question 2: Why isn't the NPC doing something else right now?

- Why weren't other Goals and/or Actions selected when we created the current Plan?

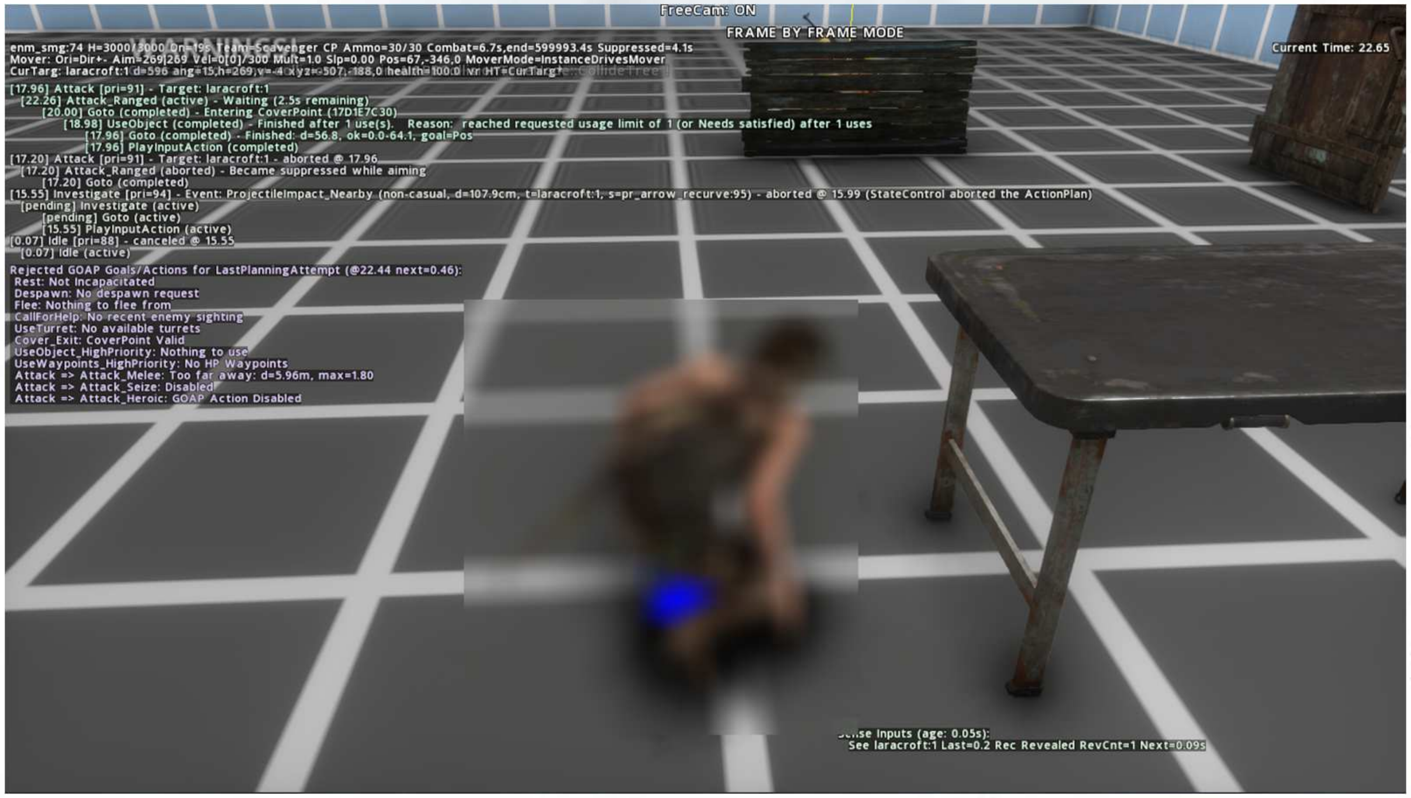
- Why haven't we been able to create a better Plan while the current Plan is active?

Common Question 3: What did he do before his current Plan, and why?

- Provide details about why other Goals/Actions weren’t selected when we created a previous Plan.

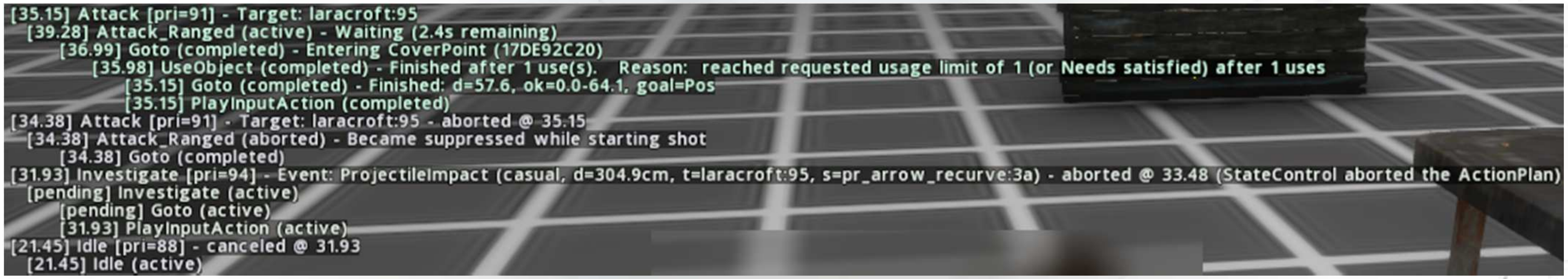
- Provide details about the final state of a previous Plan.

Tomb Raider GOAP Debugging – Last Planning Attempt

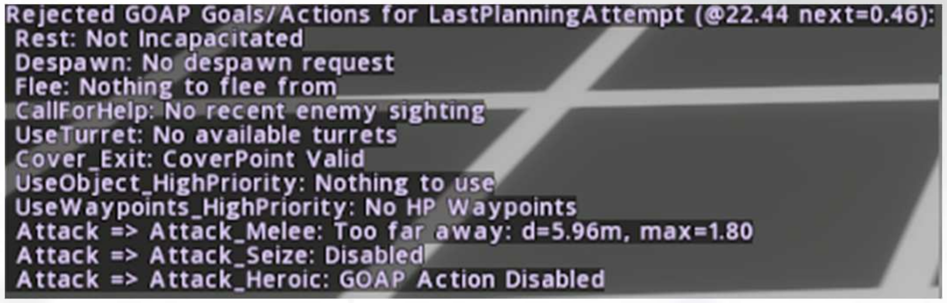


```
enm_smg:74 H=3000/3000 On=19s Team=Scavenger CP Ammo=30/30 Combat=6.7s,end=599993.4s Suppressed=4.1s
Mover: Ori=Dir+- Aim=269|269 Vel=0[0]/300 Mult=1.0 Slp=0.00 Pos=67,-346,0 MoverMode=InstanceDrivesMover
CurTarg: laracroft:1 d=596 ang=15,h=269,v=-4 xyz=-507,-188,0 health=100.0 vr HT=CurTarg

[17.96] Attack [pri=91] - Target: laracroft:1
  [22.26] Attack_Ranged (active) - Waiting (2.5s remaining)
      [20.00] Goto (completed) - Entering CoverPoint (17D1E7C30)
          [18.98] UseObject (completed) - Finished after 1 use(s).   Reason:  reached requested usage limit of 1 (or Needs satisfied) after 1 uses
              [17.96] Goto (completed) - Finished: d=56.8, ok=0.0-64.1, goal=Pos
              [17.96] PlayInputAction (completed)
[17.20] Attack [pri=91] - Target: laracroft:1 - aborted @ 17.96
  [17.20] Attack_Ranged (aborted) - Became suppressed while aiming
      [17.20] Goto (completed)
[15.55] Investigate [pri=94] - Event: ProjectileImpact_Nearby (non-casual, d=107.9cm, t=laracroft:1, s=pr_arrow_recurve:95) - aborted @ 15.99 (StateControl aborted the ActionPlan)
  [pending] Investigate (active)
      [pending] Goto (active)
      [15.55] PlayInputAction (active)
[0.07] Idle [pri=88] - canceled @ 15.55
  [0.07] Idle (active)

Rejected GOAP Goals/Actions for LastPlanningAttempt (@22.44 next=0.46):
 Rest: Not Incapacitated
 Despawn: No despawn request
 Flee: Nothing to flee from
 CallForHelp: No recent enemy sighting
 UseTurret: No available turrets
 Cover_Exit: CoverPoint Valid
 UseObject_HighPriority: Nothing to use
 UseWaypoints_HighPriority: No HP Waypoints
 Attack => Attack_Melee: Too far away: d=5.96m, max=1.80
 Attack => Attack_Seize: Disabled
 Attack => Attack_Heroic: GOAP Action Disabled

Sense Inputs (age: 0.05s):
  See laracroft:1 Last=0.2 Rec Revealed RevCnt=1 Next=0.09s
```

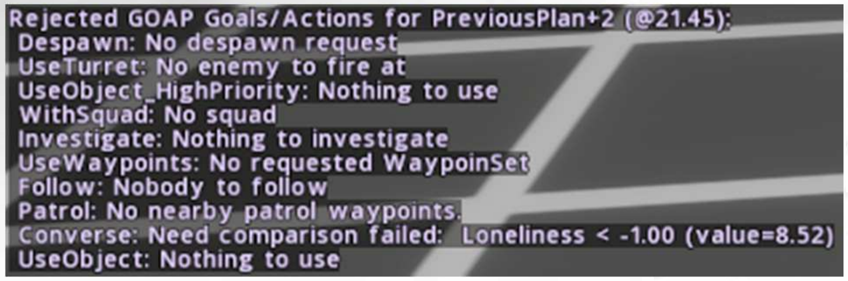
Tomb Raider GOAP Debugging – Details



```
[35.15] Attack [pri=91] - Target: laracroft:95
  [39.28] Attack_Ranged (active) - Waiting (2.4s remaining)
      [36.99] Goto (completed) - Entering CoverPoint (17DE92C20)
          [35.98] UseObject (completed) - Finished after 1 use(s).   Reason:  reached requested usage limit of 1 (or Needs satisfied) after 1 uses
              [35.15] Goto (completed) - Finished: d=57.6, ok=0.0-64.1, goal=Pos
              [35.15] PlayInputAction (completed)
[34.38] Attack [pri=91] - Target: laracroft:95 - aborted @ 35.15
  [34.38] Attack_Ranged (aborted) - Became suppressed while starting shot
      [34.38] Goto (completed)
[31.93] Investigate [pri=94] - Event: ProjectileImpact (casual, d=304.9cm, t=laracroft:95, s=pr_arrow_recurve:3a) - aborted @ 33.48 (StateControl aborted the ActionPlan)
  [pending] Investigate (active)
      [pending] Goto (active)
      [31.93] PlayInputAction (active)
[21.45] Idle [pri=88] - canceled @ 31.93
  [21.45] Idle (active)
```



```
Rejected GOAP Goals/Actions for LastPlanningAttempt (@22.44 next=0.46):
 Rest: Not Incapacitated
 Despawn: No despawn request
 Flee: Nothing to flee from
 CallForHelp: No recent enemy sighting
 UseTurret: No available turrets
 Cover_Exit: CoverPoint Valid
 UseObject_HighPriority: Nothing to use
 UseWaypoints_HighPriority: No HP Waypoints
 Attack => Attack_Melee: Too far away: d=5.96m, max=1.80
 Attack => Attack_Seize: Disabled
 Attack => Attack_Heroic: GOAP Action Disabled
```



```
Rejected GOAP Goals/Actions for PreviousPlan+2 (@21.45):
 Despawn: No despawn request
 UseTurret: No enemy to fire at
 UseObject_HighPriority: Nothing to use
 WithSquad: No squad
 Investigate: Nothing to investigate
 UseWaypoints: No requested WaypoinSet
 Follow: Nobody to follow
 Patrol: No nearby patrol waypoints.
 Converse: Need comparison failed:  Loneliness < -1.00 (value=8.52)
 UseObject: Nothing to use
```