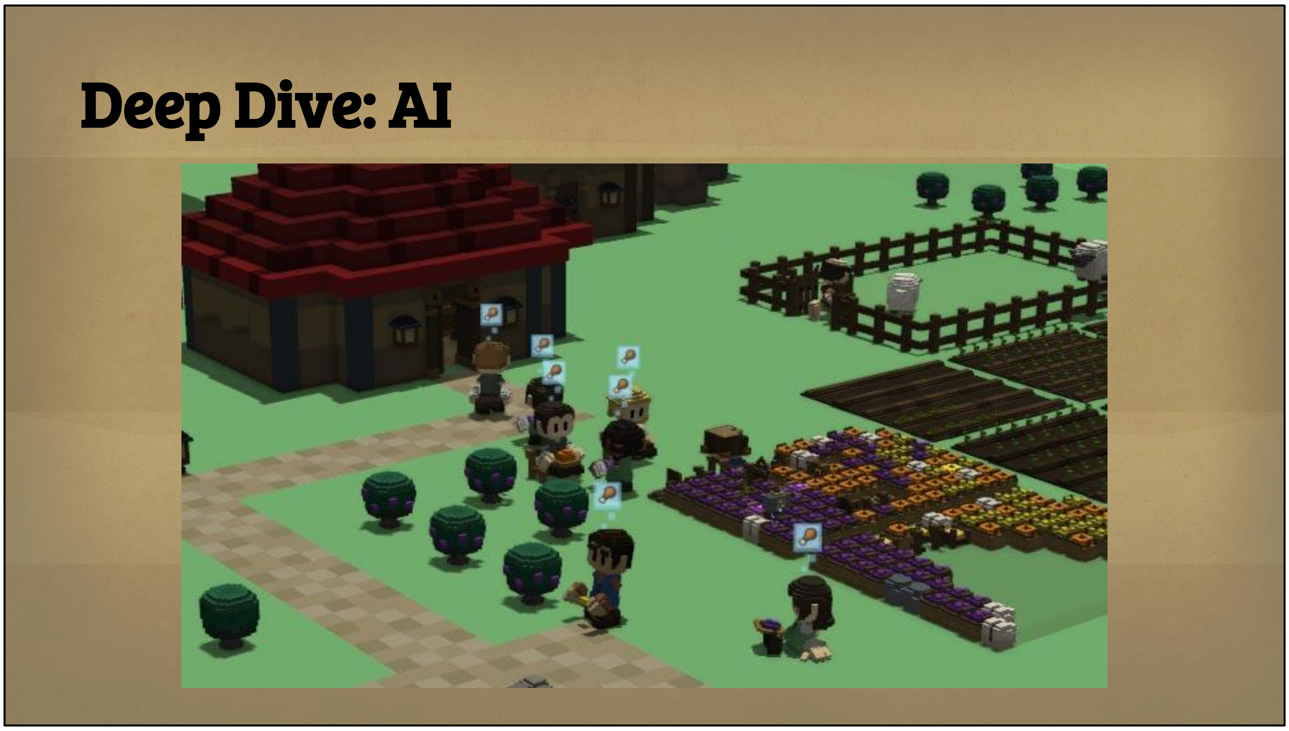
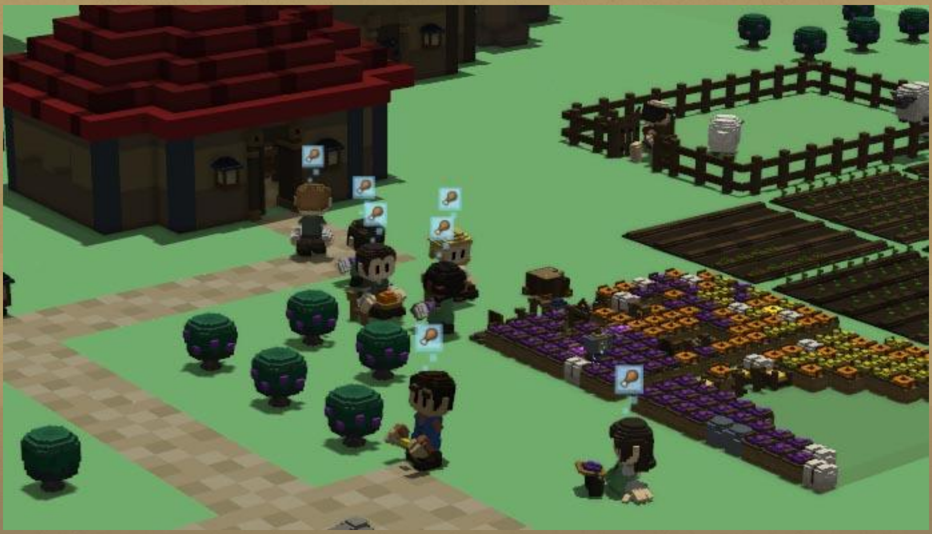


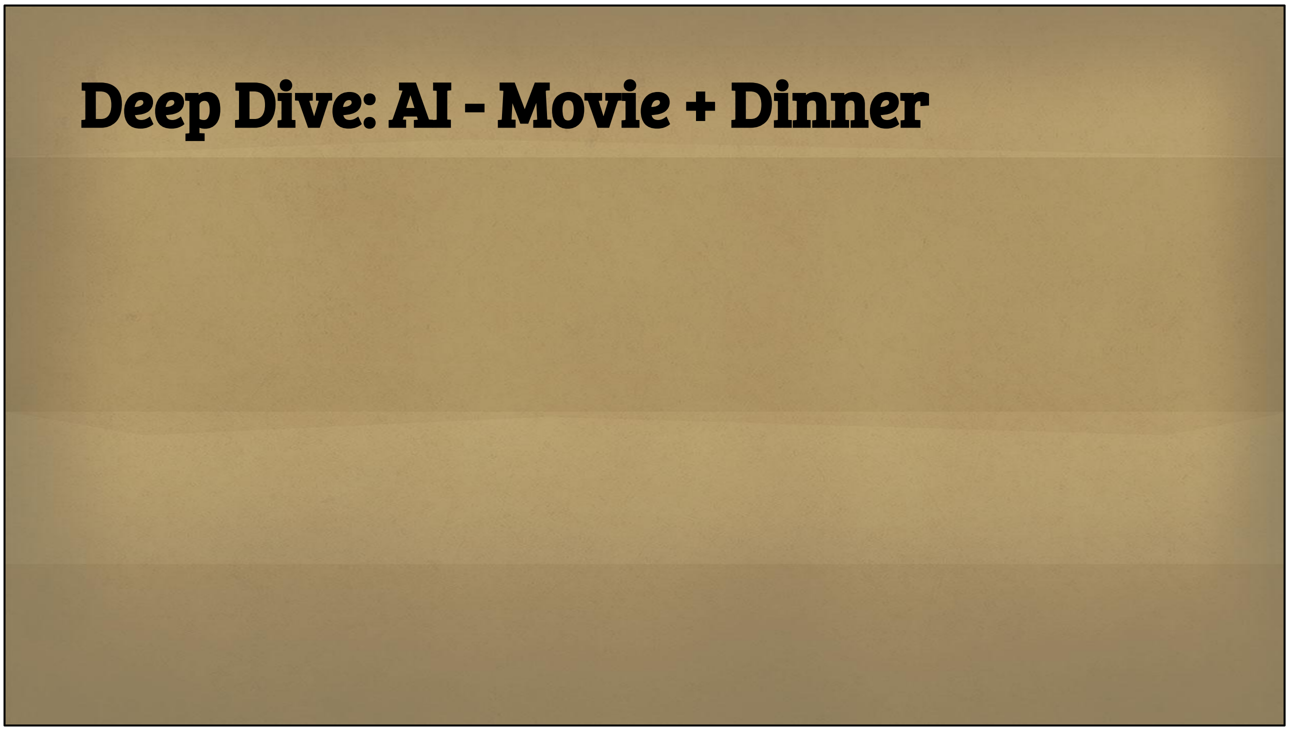
Deep Dive: AI



Today, let's take a closer look at just one part of one system: our AI.

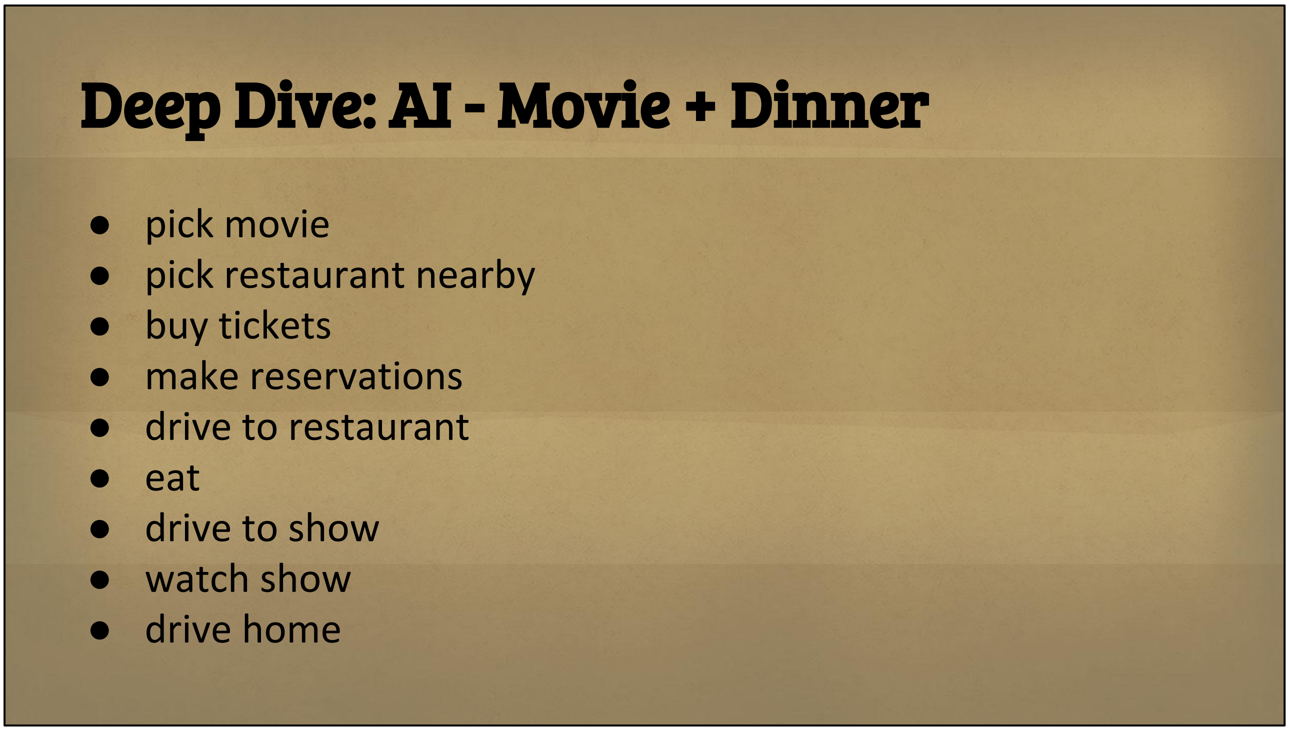
AI is a big challenge for a game like Stonehearth. In the screen here, you can see that at lunchtime, everyone eats at the same time. This involves looking for food to eat, picking it up, finding somewhere to sit, and then eating the food. Also, repeating this until the person stops being hungry. Can you think why this might be complicated?

Right, so because SH is a town-level game, all the people in the town need to do things, and each of their tasks must be broken down such that all possible points of overlap and contact are coordinated. For example, two people can't sit in the same chair, or eat the same food.



Deep Dive: AI - Movie + Dinner

Let’s do a Real Life example to illustrate the complexity. This is actually the real-life example we used when we were working out how to make the AI system. Let's say that you want to go see a movie and then also get dinner somewhere nearby, beforehand. What sorts of things do you have to do, in order to accomplish this task?

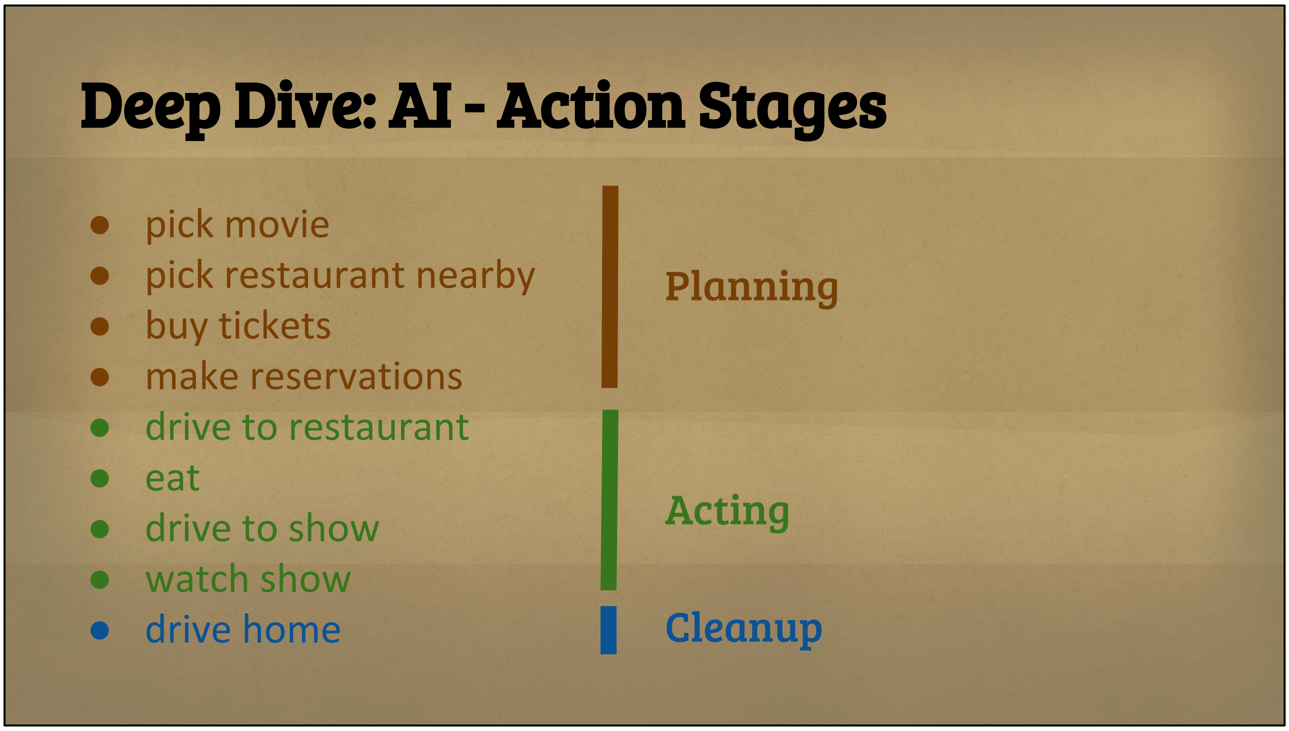


Deep Dive: AI - Movie + Dinner

- pick movie
- pick restaurant nearby
- buy tickets
- make reservations
- drive to restaurant
- eat
- drive to show
- watch show
- drive home

Right, so first before you go see the movie, you have to pick the movie. Based on the movie you pick, that narrows down the possible theaters you can go to and the times that you'll be seeing it. This in turn narrows down the restaurants available to you. You pick a restaurant based on these constraints. Then comes the verification step: can you in fact, in advance, through your phone, buy tickets for this movie, and make reservations for the restaurant? If you can neither buy tickets or get a reservation, you may have to redo one or both of the picking steps--either pick a different movie or pick a different restaurant. Many permutations.

Once you have decided on the movie and restaurant, you can then execute your evening--drive to the restaurant, eat, drive to the movie, watch it, and go home.



Deep Dive: AI - Action Stages

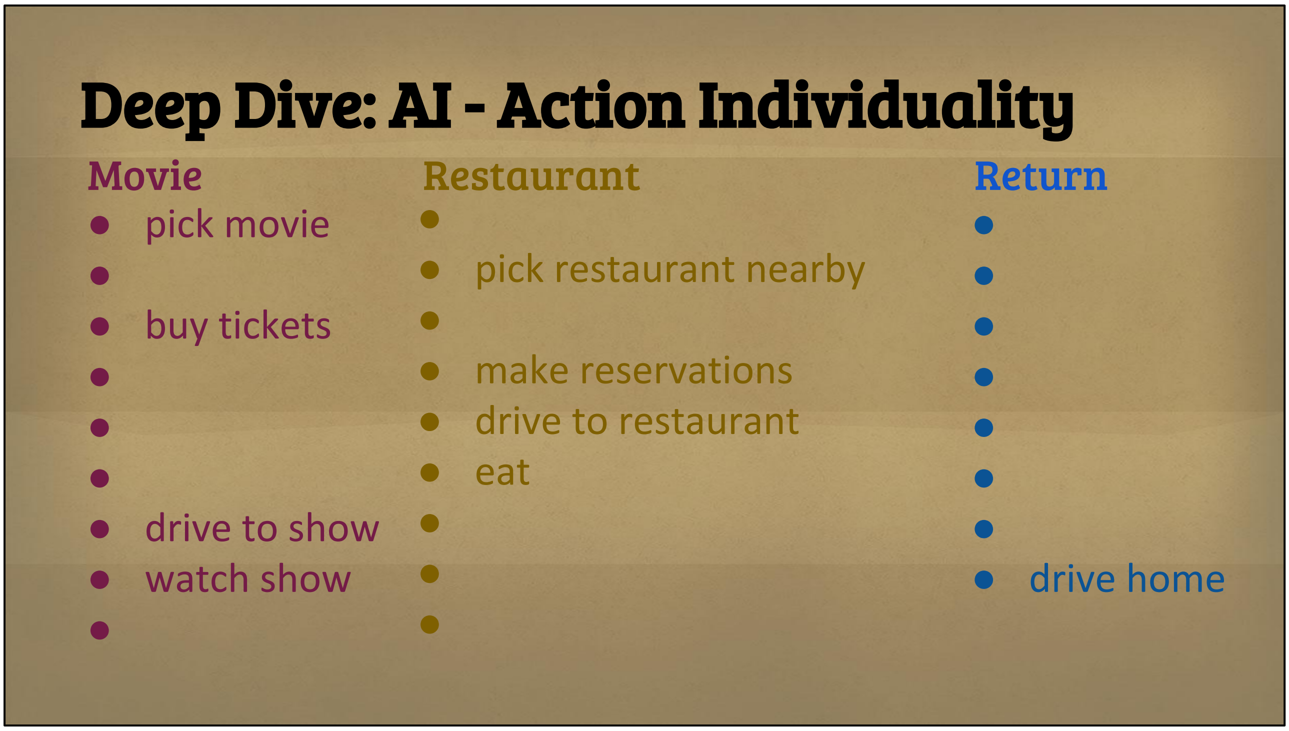
- pick movie
- pick restaurant nearby
- buy tickets
- make reservations
- drive to restaurant
- eat
- drive to show
- watch show
- drive home

Planning

Acting

Cleanup

This breakdown means that there are 3 distinctly different parts to the activities involved in this movie+restaurant activity: the planning, part, the acting part, and the cleanup/resetting state part. This breakdown is useful because it lets us isolate the planning form the acting, so that we don't have to get halfway through the evening, and then change plans. If this were a program, an AI, we could write it down just like this, where each action is a function, and call it a day right?



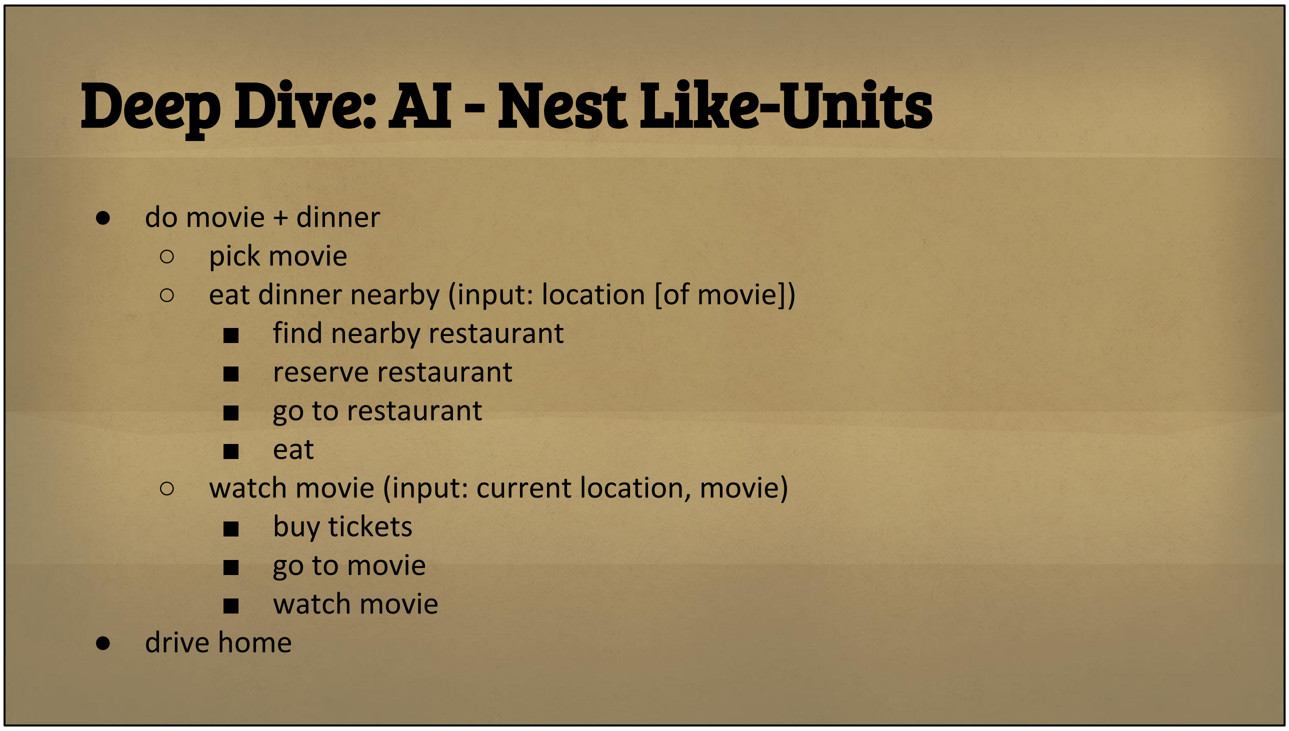
Deep Dive: AI - Action Individuality

Movie	Restaurant	Return
pick movie	●	●
●	pick restaurant nearby	●
buy tickets	●	●
●	make reservations	●
●	drive to restaurant	●
●	eat	●
drive to show	●	●
watch show	●	drive home
●	●	

Well, yes we could, but planning/action isn't the only way to divide up the steps. Here… you can see that this is another way we'd divide up/categorize the steps. We can also separate everything involved with the movie, from everything involved with the restaurant, from everything involved with the driving home. Can anyone think which might be better/more useful and why?

Well, as any good engineer knows, you also usually want to write your code in such a way that like-functions/behaviors stay together, so that you can re-use your code. What if I wanted to write an AI in which I visited only a restaurant, or only went to a movie? What if I wanted to write it so that I went to a restaurant and then ice skating? Dividing the actions up this way would allow me to do that, at the expense of having actions that can be easily interleaved and dependent on each other.

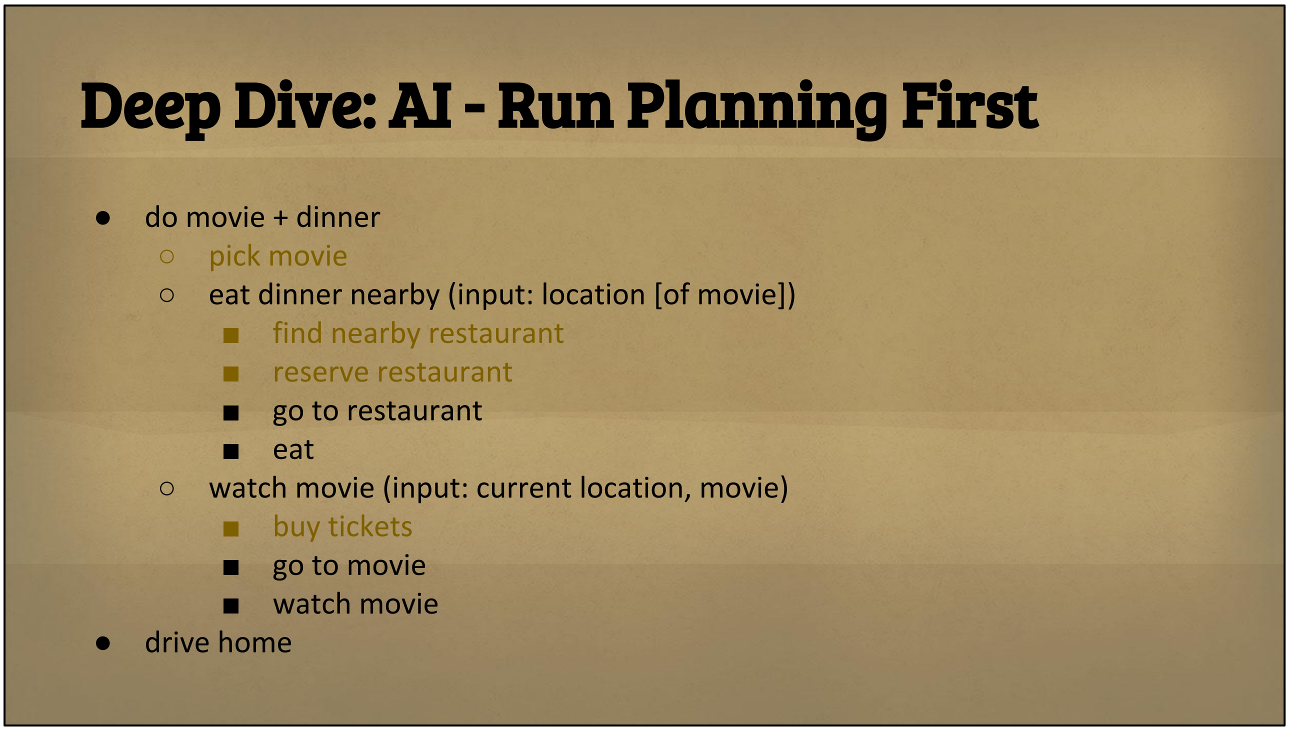
So what's the solution? How do I get interdependence but also re-usabiity?



Deep Dive: AI - Nest Like-Units

- do movie + dinner
  - pick movie
  - eat dinner nearby (input: location [of movie])
    - find nearby restaurant
    - reserve restaurant
    - go to restaurant
    - eat
  - watch movie (input: current location, movie)
    - buy tickets
    - go to movie
    - watch movie
- drive home

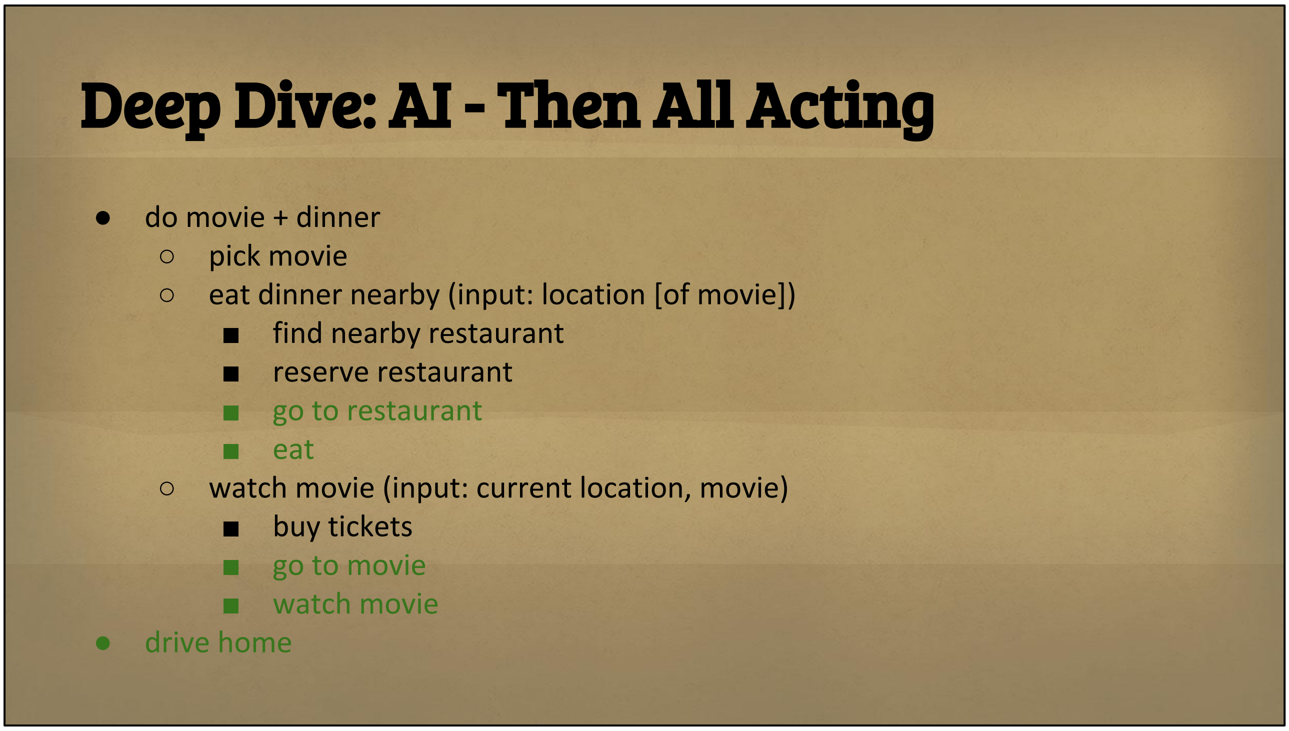
Solution: nest like units, so that the primary action is movie + dinner, and then inside that action there is a dinner section, and a movie section… each of which takes as inputs, the outcome of the planning session before.



Deep Dive: AI - Run Planning First

- do movie + dinner
    - pick movie
    - eat dinner nearby (input: location [of movie])
        - find nearby restaurant
        - reserve restaurant
        - go to restaurant
        - eat
    - watch movie (input: current location, movie)
        - buy tickets
        - go to movie
        - watch movie
- drive home

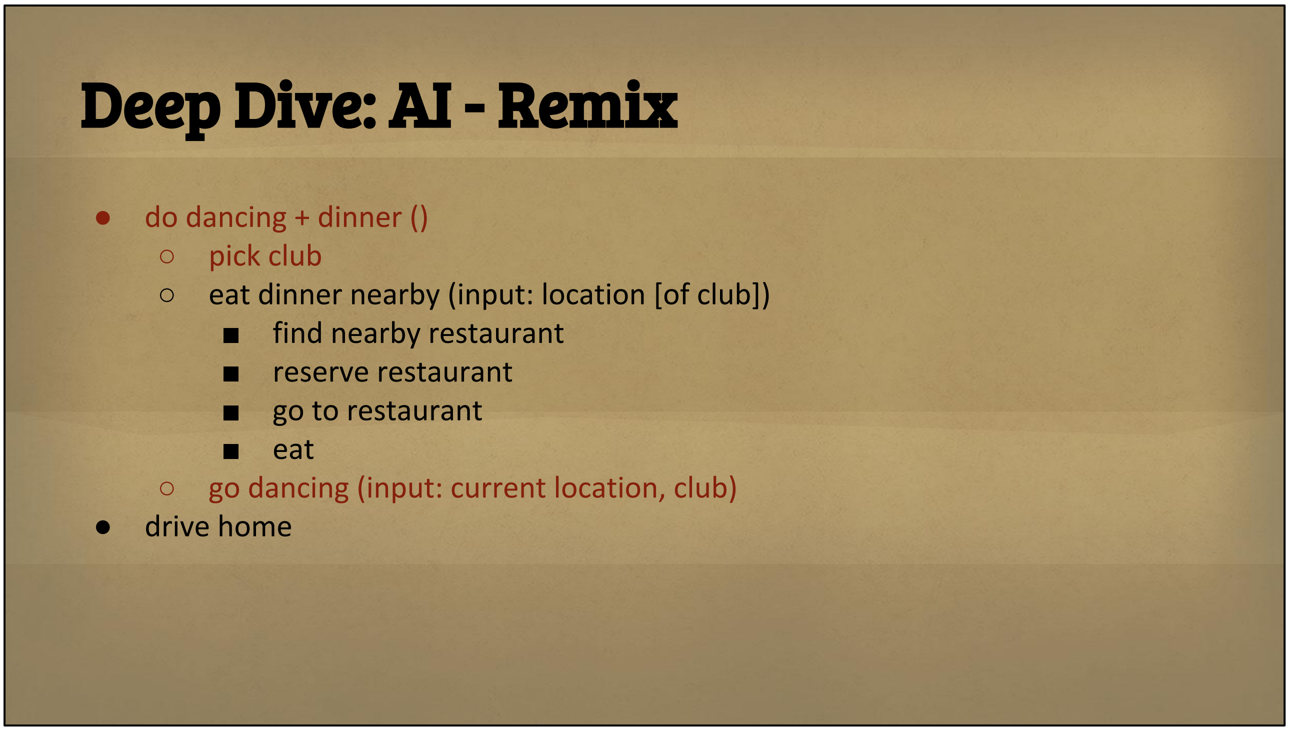
Then run all the planning actions *first* and in order. If any fail, start the whole stack again. So first: pick a movie. Use this as input to the "find nearby restaurant." Given a restaurant, then then do "reserve restaurant". If reserve fails, start that unit over, meaning, find another restaurant. If the unit fails, then find another movie. If the reserve succeeds, go to "buy tickets". Ïf buy tickets fails, start the stack over, meaning, find another movie.



Deep Dive: AI - Then All Acting

- do movie + dinner
  - pick movie
  - eat dinner nearby (input: location [of movie])
    - find nearby restaurant
    - reserve restaurant
    - go to restaurant
    - eat
  - watch movie (input: current location, movie)
    - buy tickets
    - go to movie
    - watch movie
- drive home

Then, only if all the planning stages succeed, do you do all the execution actions in order. But you know these won't fail, because the reservations/tickets are already there for you.

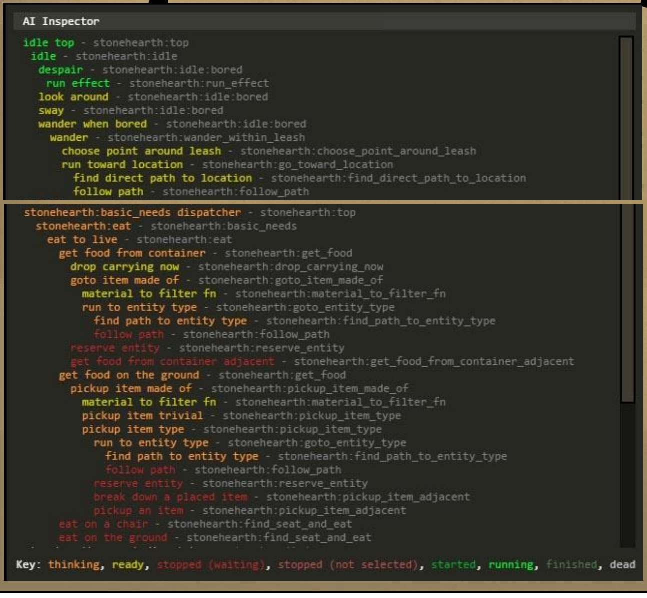


Deep Dive: AI - Remix

- do dancing + dinner ()
    - pick club
    - eat dinner nearby (input: location [of club])
        - find nearby restaurant
        - reserve restaurant
        - go to restaurant
        - eat
    - go dancing (input: current location, club)
- drive home

This way you can remix things--if you want to reuse the eat dinner action to do dinner + dancing, you don't have to write an all-new function, you just have to write the dancing part. Win!

Deep Dive: AI - In the game

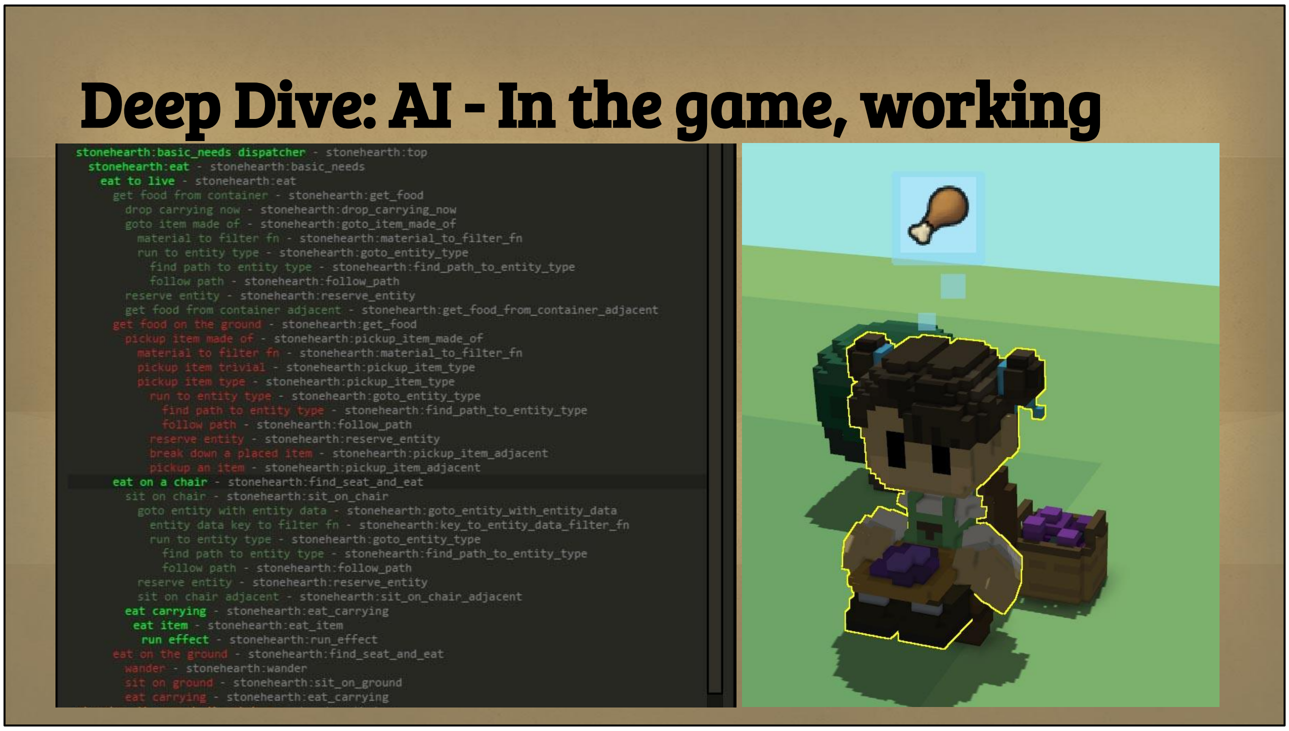


```
AI Inspector
idle top - stonehearth:top
 idle - stonehearth:idle
  despair - stonehearth:idle:bored
   run effect - stonehearth:run_effect
  look around - stonehearth:idle:bored
  sway - stonehearth:idle:bored
  wander when bored - stonehearth:idle:bored
   wander - stonehearth:wander_within_leash
    choose point around leash - stonehearth:choose_point_around_leash
    run toward location - stonehearth:go_toward_location
     find direct path to location - stonehearth:find_direct_path_to_location
     follow path - stonehearth:follow_path

stonehearth:basic_needs dispatcher - stonehearth:top
 stonehearth:eat - stonehearth:basic_needs
  eat to live - stonehearth:eat
   get food from container - stonehearth:get_food
    drop carrying now - stonehearth:drop_carrying_now
    goto item made of - stonehearth:goto_item_made_of
     material to filter fn - stonehearth:material_to_filter_fn
     run to entity type - stonehearth:goto_entity_type
      find path to entity type - stonehearth:find_path_to_entity_type
      follow path - stonehearth:follow_path
    reserve entity - stonehearth:reserve_entity
    get food from container adjacent - stonehearth:get_food_from_container_adjacent
   get food on the ground - stonehearth:get_food
    pickup item made of - stonehearth:pickup_item_made_of
     material to filter fn - stonehearth:material_to_filter_fn
     pickup item trivial - stonehearth:pickup_item_type
     pickup item type - stonehearth:pickup_item_type
      run to entity type - stonehearth:goto_entity_type
       find path to entity type - stonehearth:find_path_to_entity_type
       follow path - stonehearth:follow_path
      reserve entity - stonehearth:reserve_entity
      break down a placed item - stonehearth:pickup_item_adjacent
      pickup an item - stonehearth:pickup_item_adjacent
   eat on a chair - stonehearth:find_seat_and_eat
   eat on the ground - stonehearth:find_seat_and_eat
Key: thinking, ready, stopped (waiting), stopped (not selected), started, running, finished, dead
```

And this is what it looks like in the game. The dev version of the game has an AI inspector showing all the things that could be running on a person at a given time.

At the top you can see that though the person hungry, she's running idle->despair (current actions are in green) because simultaneously, she's trying to run eat, but it's getting stuck in the thinking/planning part: she can't find any food to pick up. Since she can't find any food in follow path, she can't reserve it and the running to the food will never happen.
--
(It's a little hard to read because vertical alignment means both 'peer actions--all the equivalent things I could be doing', and sequential actions within an action, or all the things I need to do to make this happen, but let's just focus on one action at a time.)



Deep Dive: AI - In the game, working

```
stonehearth:basic_needs dispatcher - stonehearth:top
 stonehearth:eat - stonehearth:basic_needs
  eat to live - stonehearth:eat
   get food from container - stonehearth:get_food
    drop carrying now - stonehearth:drop_carrying_now
    goto item made of - stonehearth:goto_item_made_of
     material to filter fn - stonehearth:material_to_filter_fn
     run to entity type - stonehearth:goto_entity_type
      find path to entity type - stonehearth:find_path_to_entity_type
      follow path - stonehearth:follow_path
    reserve entity - stonehearth:reserve_entity
    get food from container adjacent - stonehearth:get_food_from_container_adjacent
   get food on the ground - stonehearth:get_food
    pickup item made of - stonehearth:pickup_item_made_of
     material to filter fn - stonehearth:material_to_filter_fn
     pickup item trivial - stonehearth:pickup_item_type
     pickup item type - stonehearth:pickup_item_type
      run to entity type - stonehearth:goto_entity_type
       find path to entity type - stonehearth:find_path_to_entity_type
       follow path - stonehearth:follow_path
      reserve entity - stonehearth:reserve_entity
      break down a placed item - stonehearth:pickup_item_adjacent
      pickup an item - stonehearth:pickup_item_adjacent
   eat on a chair - stonehearth:find_seat_and_eat
    sit on chair - stonehearth:sit_on_chair
     goto entity with entity data - stonehearth:goto_entity_with_entity_data
      entity data key to filter fn - stonehearth:key_to_entity_data_filter_fn
      run to entity type - stonehearth:goto_entity_type
       find path to entity type - stonehearth:find_path_to_entity_type
       follow path - stonehearth:follow_path
     reserve entity - stonehearth:reserve_entity
     sit on chair adjacent - stonehearth:sit_on_chair_adjacent
    eat carrying - stonehearth:eat_carrying
     eat item - stonehearth:eat_item
      run effect - stonehearth:run_effect
   eat on the ground - stonehearth:find_seat_and_eat
    wander - stonehearth:wander
    sit on ground - stonehearth:sit_on_ground
    eat carrying - stonehearth:eat_carrying
```

Once there is food in the world, she is able to locate and reserve it, and the planning section (thinking section) finishes, allowing the eating action to actually run. (In our dinner/movie analogy, she's found the tickets and the reservation and is now eating/watching the movie.) The green means that the planning stage for the actions mentioned action have succeeded, and the AI can progress.

As you can see, actions have an incredibly fine amount of detail. The reason I used the dinner/movie analogy is because it is ONLY at an hour-long activity level that humans start to think that we need to break our behavior down, and plan everything in advance. We actually do the same when we're trying to get a seat at a buffet, but it's a much more automatic process. In Stonehearth, however, and all similar kinds of programming problems, the AI has to handle each little interaction as if each were a big deal.