Faculty of Mathematics and Physics
Charles University
1st March 2023

UT2004 bots made easy!

Pogamut 3

Lab 03 – Navigation, Items, Communication

Warm Up!

- Fill the short test for this lessons

  6 minutes limit

  https://tinyurl.com/42sk3nxw

  `O vs. 0, i vs. l vs. 1`

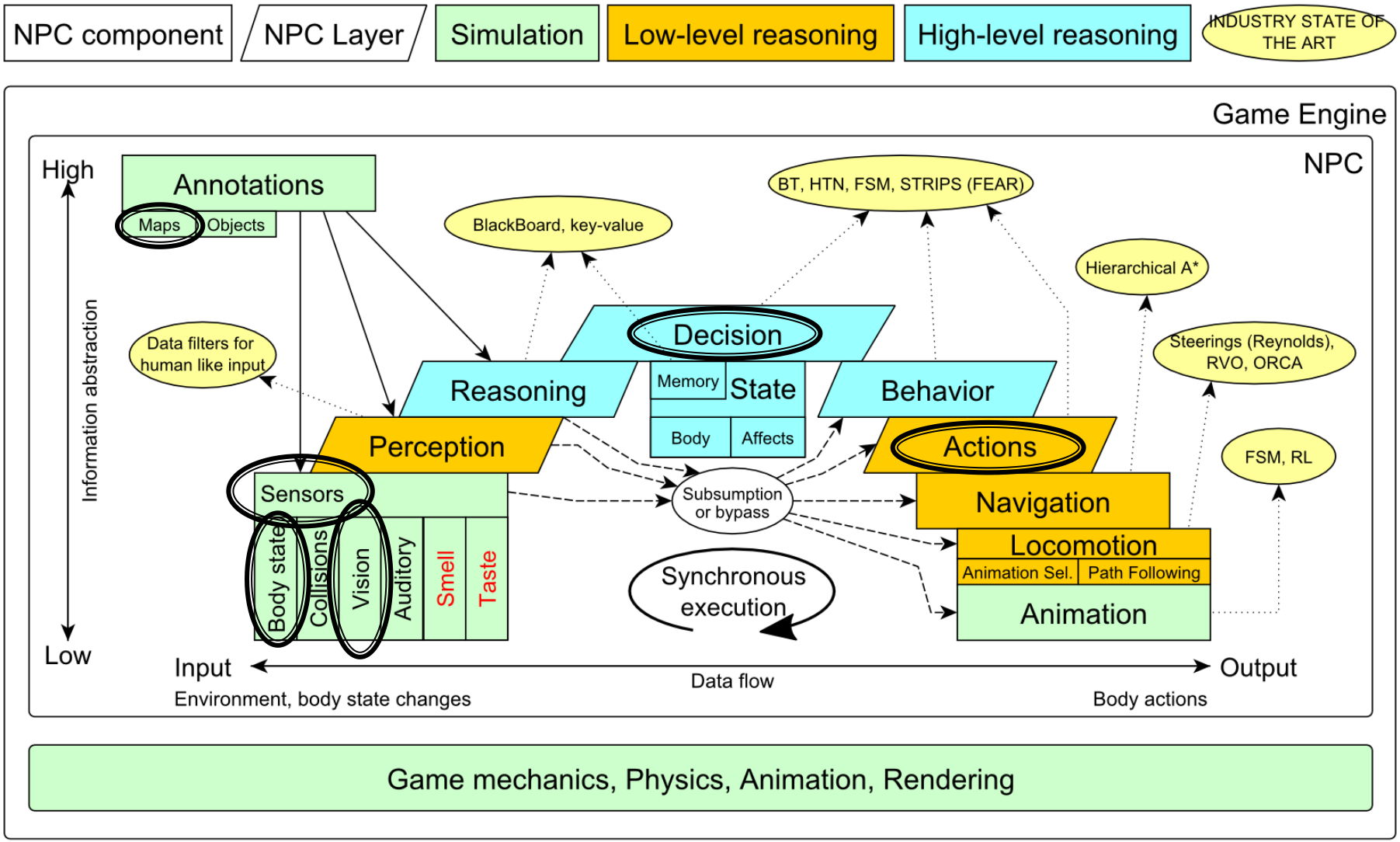
  Permalink

  https://docs.google.com/forms/d/e/1FAIpQLSfSbAPWPbwqY16dq8dLOsYtbIiyeh9zomenpMvK4nW4hutA_w/viewform

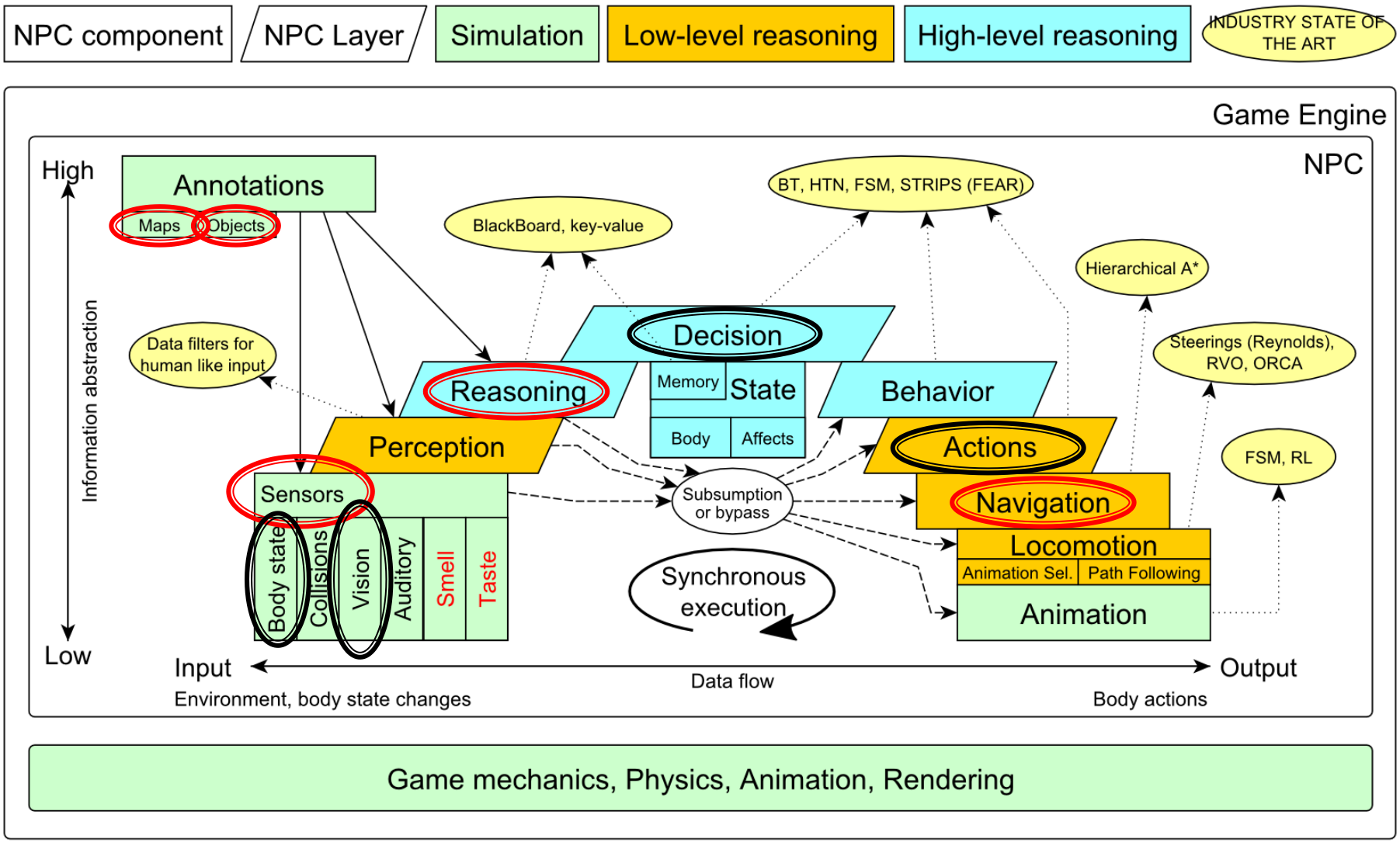
Today's menu

1. **Big Picture**
2. ItemPickerSquad Homework
3. World Abstraction
4. Navigation Environment Abstraction
5. Items
6. Navigation
7. Team Communication

Big Picture
Already covered



Big Picture
Today



Homework 03

ItemPicker Squad

3 bots, many items to pick, time is ticking

Homework 03
Gotta COLLECT 'em all!

- You have to implement an `ItemPickerBot` squad that will be able to collect all "interesting" items on the map; **3 bots** in a team
- You will have to communicate **as if you are not running in the same JVM**
- 3 maps to play on
  - DM-1on1-Albatross, DM-Rankin-FE, DM-1on1-Roughinery-FPS
- Interesting items: **Weapons, Armors, Shields**
- BASE: implement a picking squad that shares info about items they picked
- ADVANCED: **make sure** that a situation "two bots are pursuing the same item" never happens + if two bots want the same item, then one that is nearer will pursue it
  - 10 advanced points
  - Deadline for advanced points: see the website
- Squad template (includes UT2004 patch)
  - https://tinyurl.com/77a3uewt
  - Full link: https://drive.google.com/file/d/1qsJp3-HXtWzLCkK9liSUfhOeJu2KwzKN/view?usp=sharing

World Abstraction

`Item, Navpoint`

+ modules, helper classes and events

Pogamut 3 World Abstraction

New Stuff

Objects - `IWorldObject`

- `Player`
- `Self`
- *New* `Item`
- *New* `NavPoint`

Events - `IWorldEvent`

- `GlobalChat`
- *New* `HearPickup`
  - I've heard someone else
- *New* `ItemPickedUp`
  - I've picked up something

- Modules:
  - this.players, this.info
  - *New* this.items, this.navPoints
- *New* Helper classes
  - `MyCollections` – operations over collections
  - `DistanceUtils` – operations over distances
- Registering `IWorldEvent` listener example

```
@EventListener(eventClass = ItemPickedUp.class)
public void pickedUp(ItemPickedUp chatEvent) { … }
```

You can register callback to any class that is implementing `IWorldEvent` using this template.

Environment Abstraction

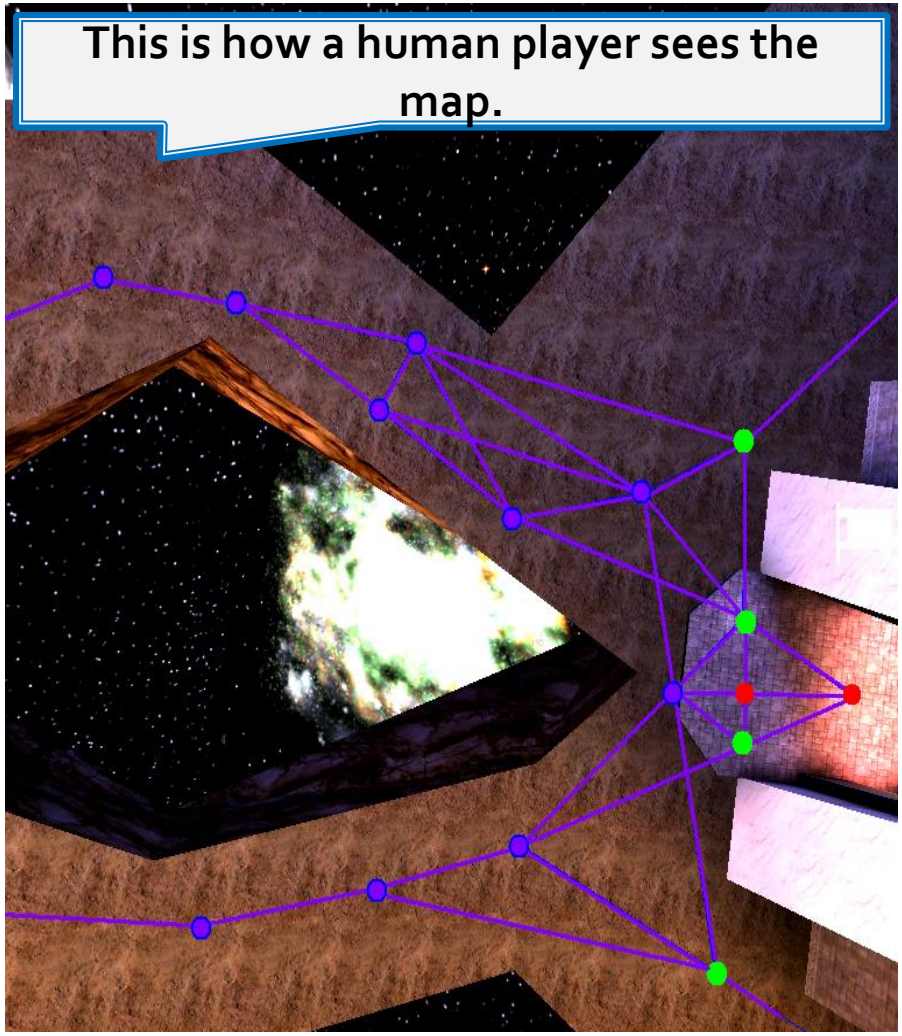
Navigation Graph

+

Navigation Mesh

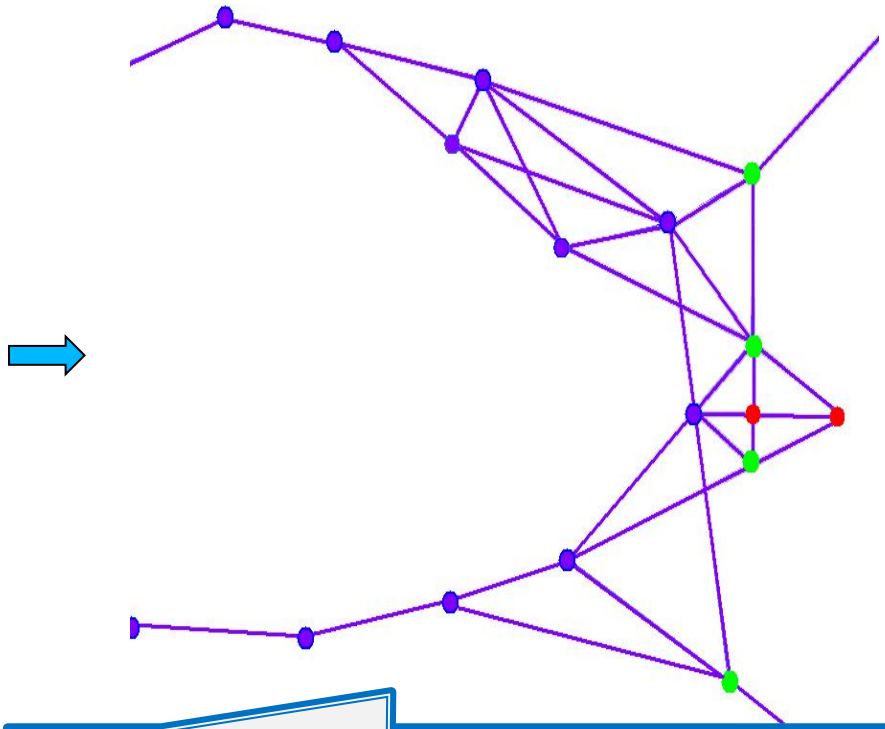
Navigation Graph

Original UT2004 navigation abstraction



This is how original UT2004 bots "see" this map.

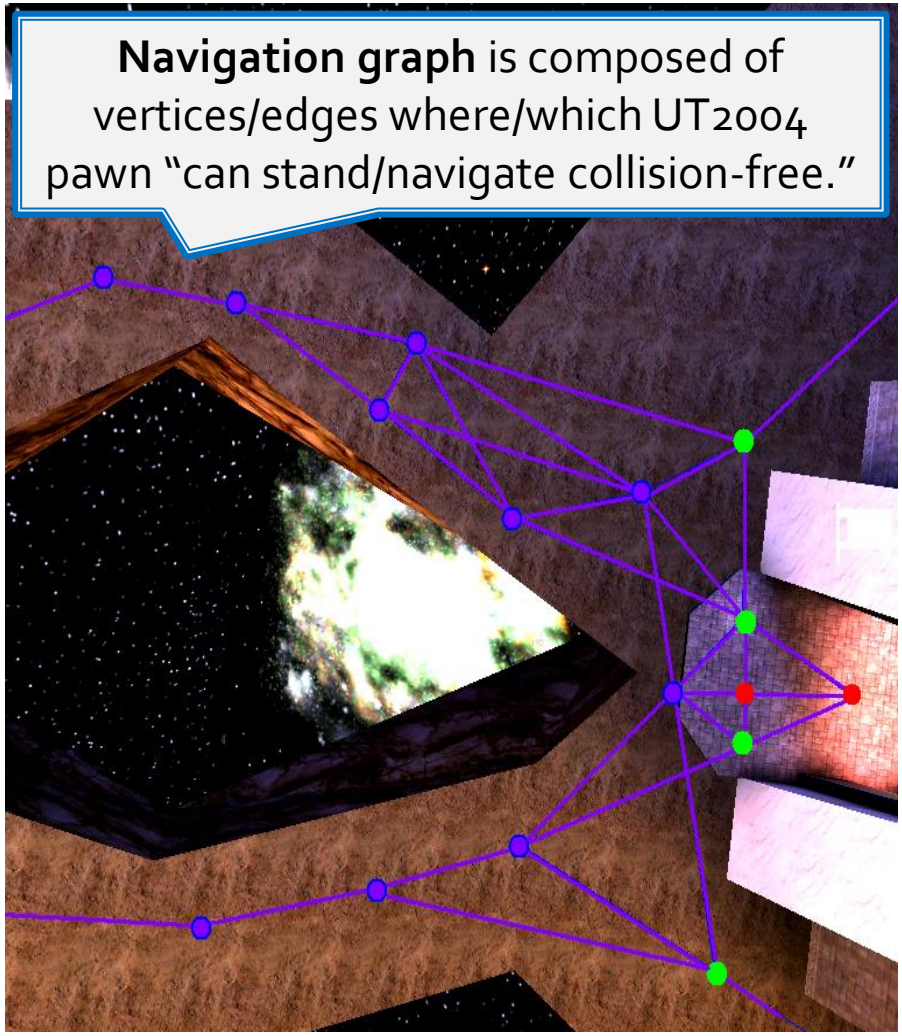
#Navpoints in a map = 100 – 5000



This is how the bot sees the environment!

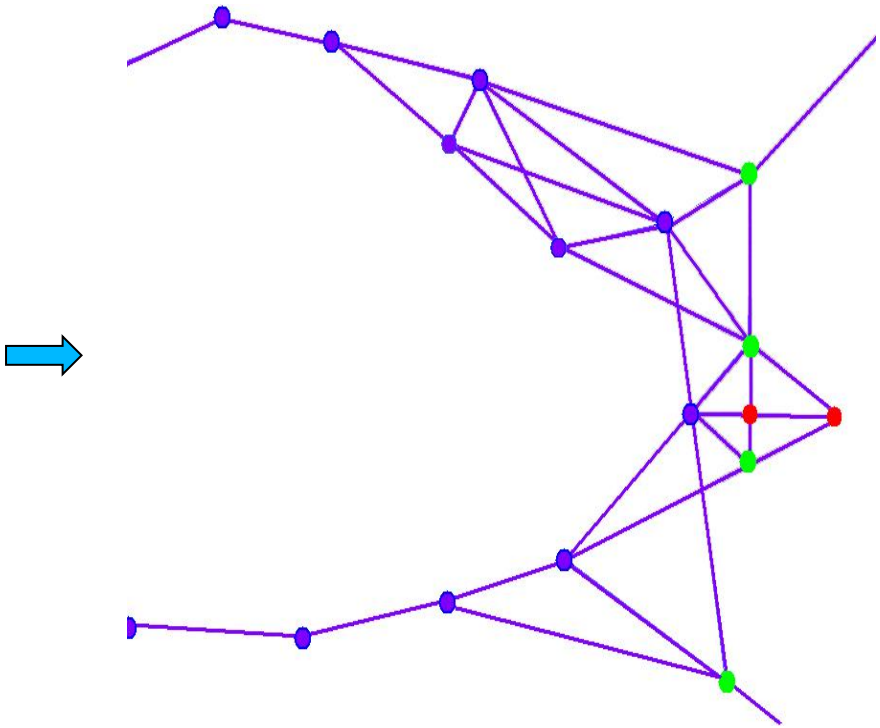
Navigation Graph

Original UT2004 navigation abstraction



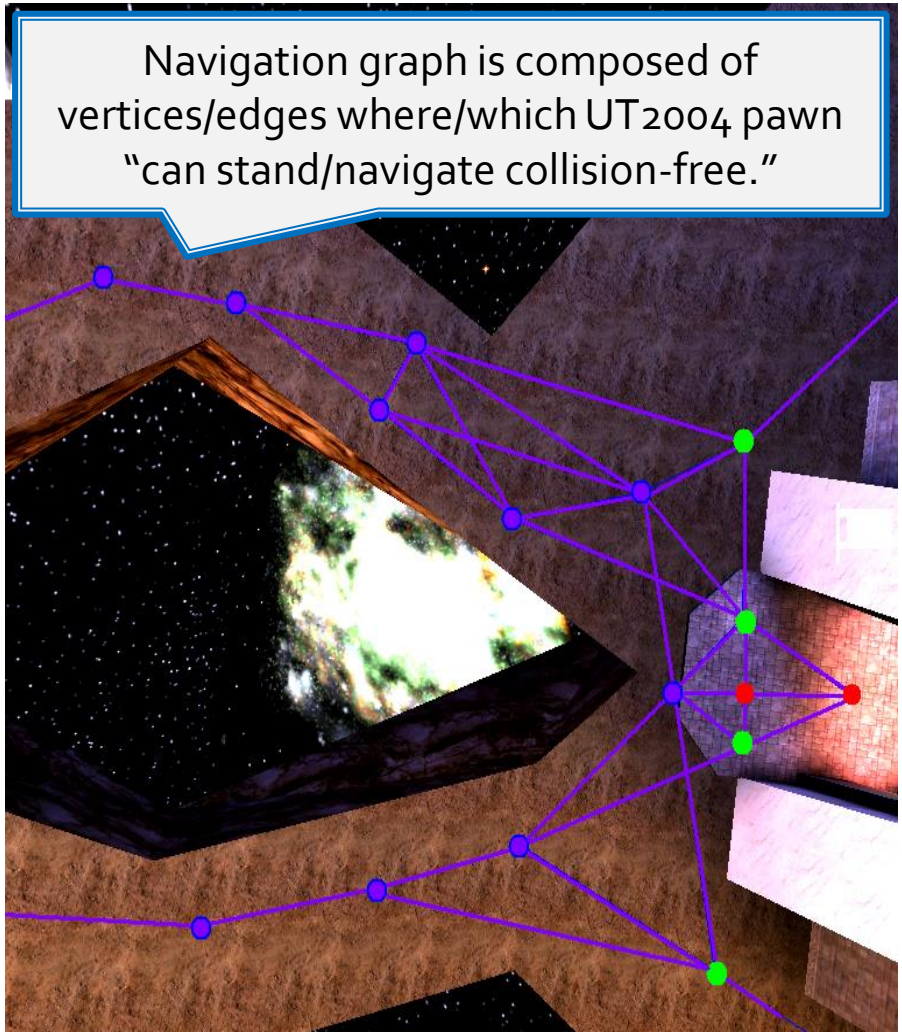
This is how original UT2004 bots "see" this map.

#Navpoints in a map = 100 – 5000



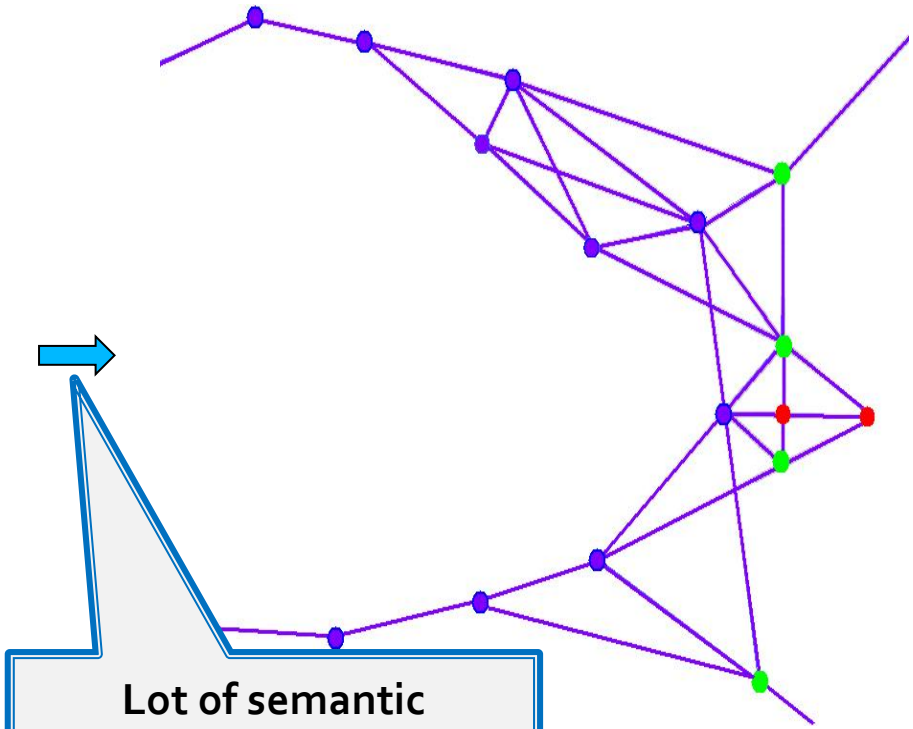
Navigation Graph

Original UT2004 navigation abstraction



This is how original UT2004 bots "see" this map.

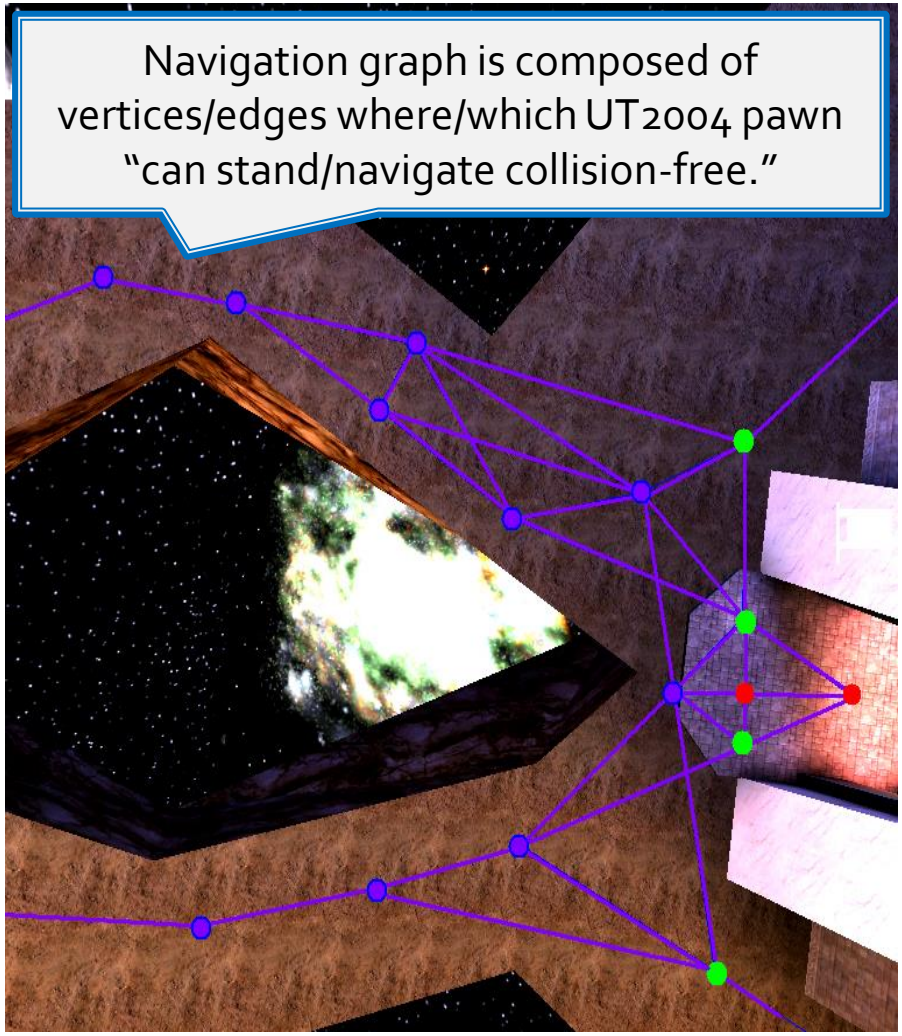
#Navpoints in a map = 100 – 5000



Lot of semantic information is lost (e.g. walls, holes, visibility)!

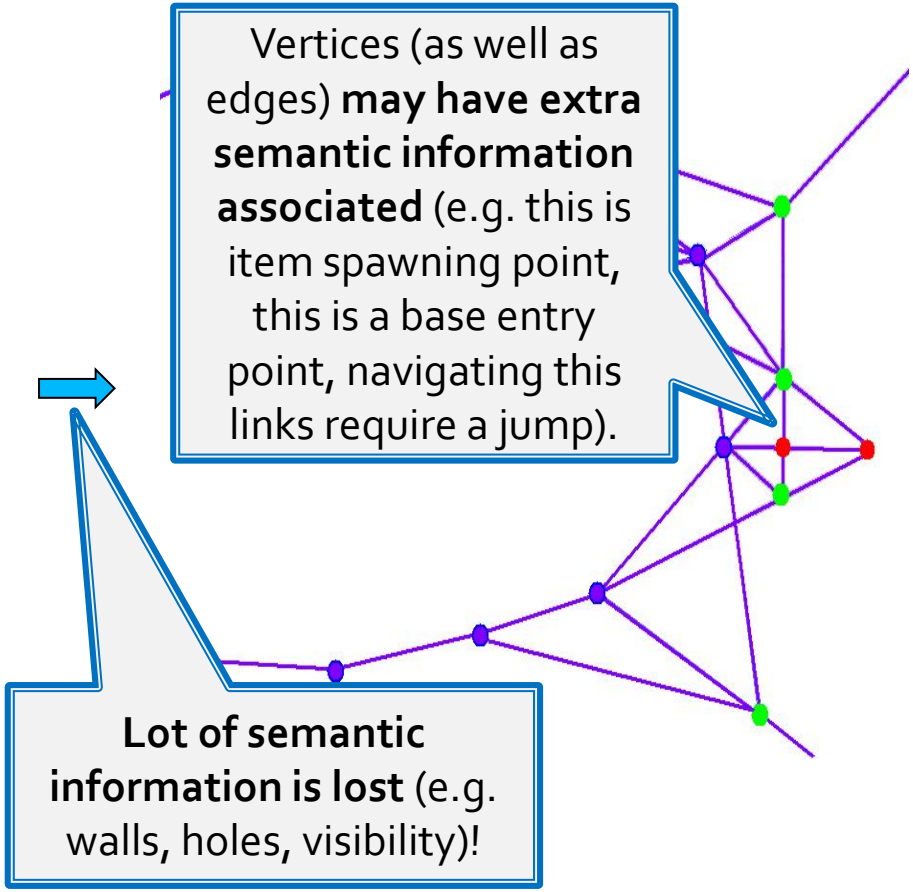
Navigation Graph

Original UT2004 navigation abstraction



This is how original UT2004 bots "see" this map.

#Navpoints in a map = 100 – 5000



Navigation Graph

Low-level API

Classes of interest:

`NavPoints` – module wrapping navpoints

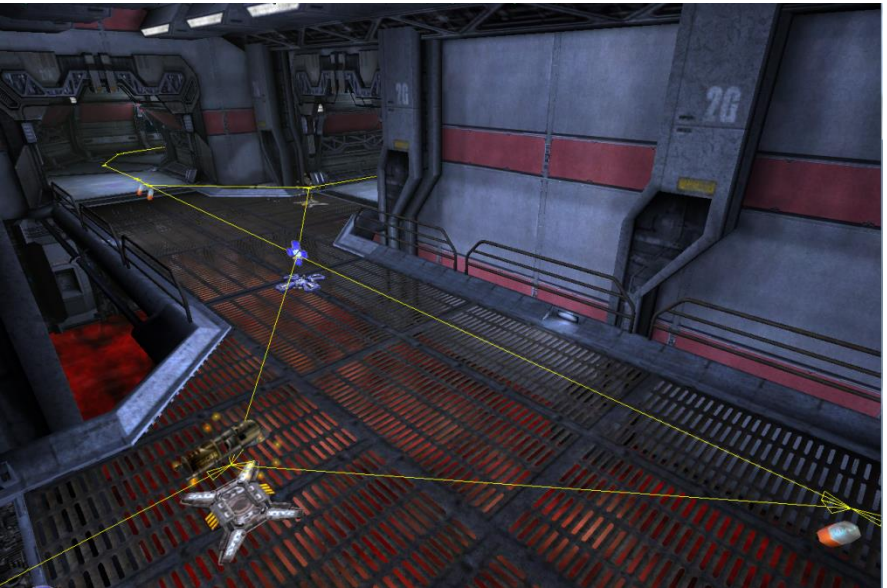
`NavPoint`, `NavPointNeighbourLink` – navgraph data

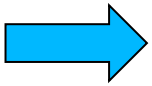
`Item` == usually item spawning point!

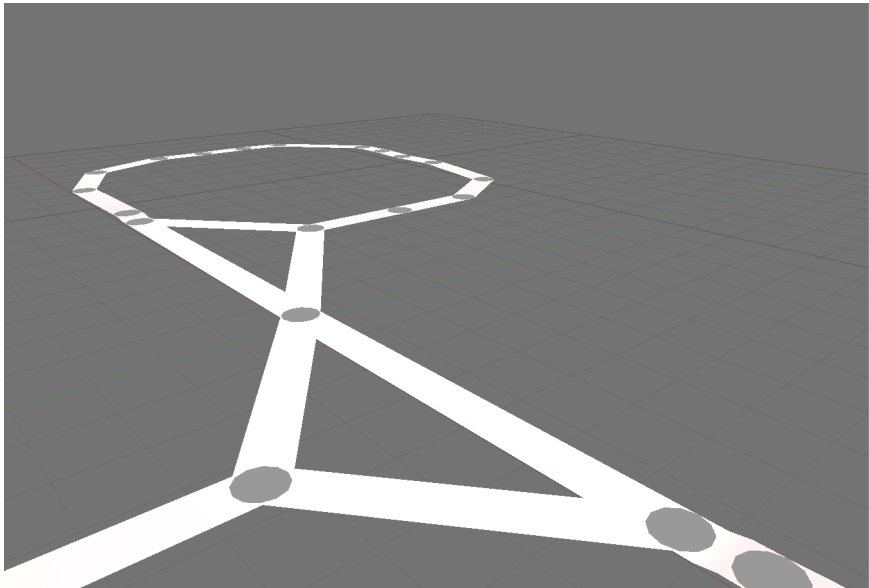
`NavigationGraphBuilder`

Methods of interest:

```
this.navPoints.getNavPoints()
this.items.getAllItems ()
this.world.getAll(NavPoint.class)
this.world.getAll(Item.class)
someNavPoint.getOutgoingEdges()
someNavPoint.getIncomingEdges()
```





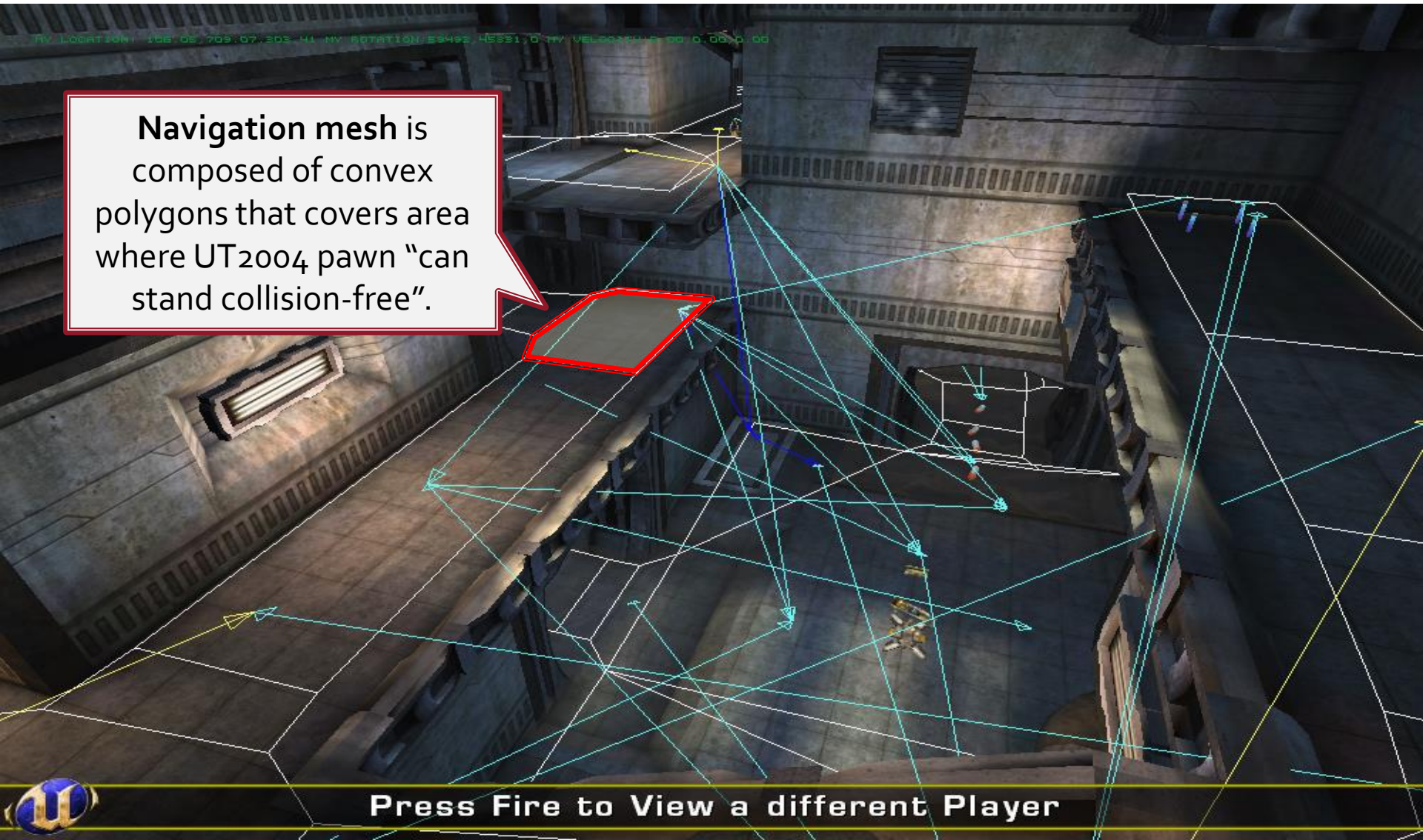


Navigation Mesh

Custom navig. abstr. computed from the map geom.

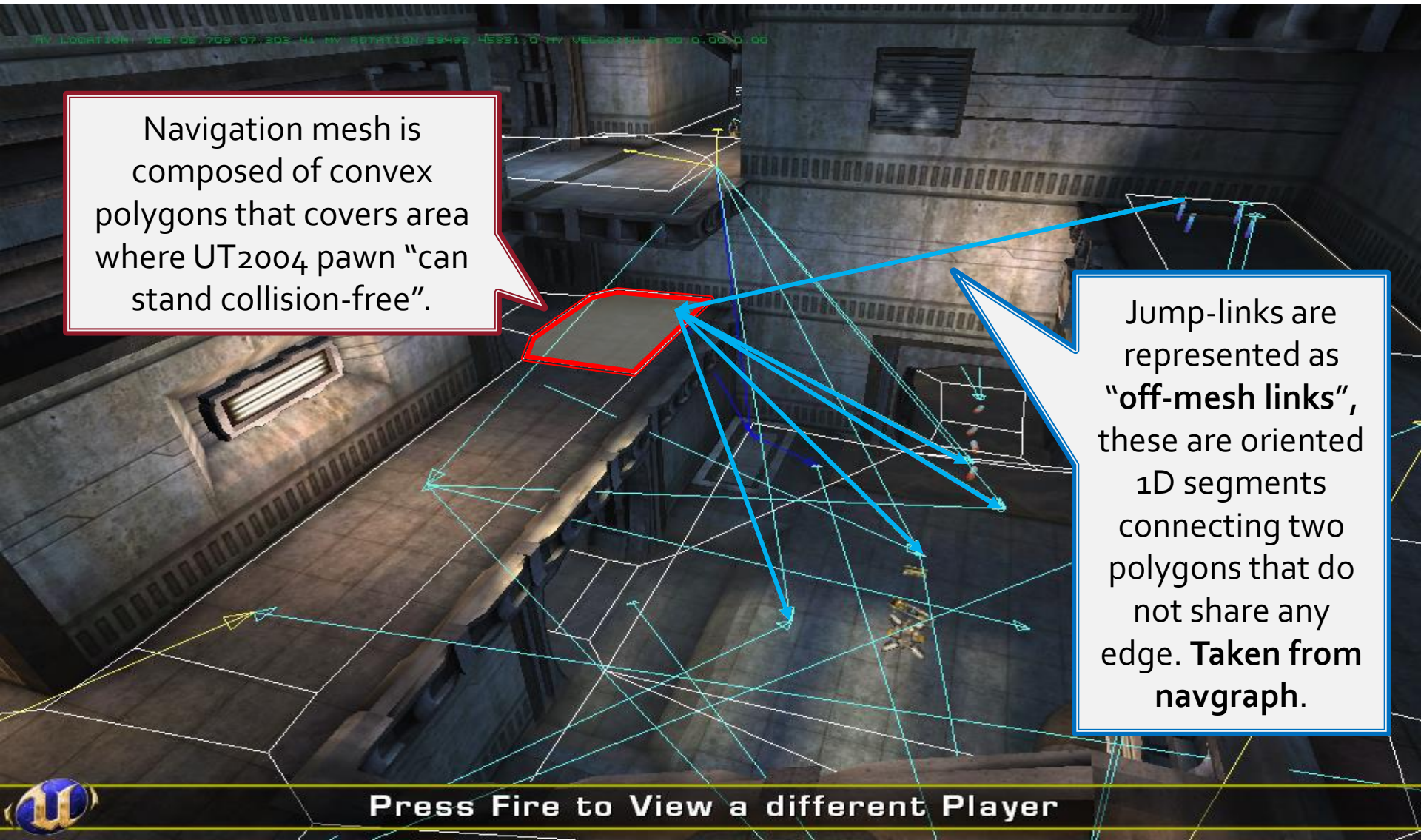
Navigation Mesh

Custom navig. abstr. computed from the map geom.



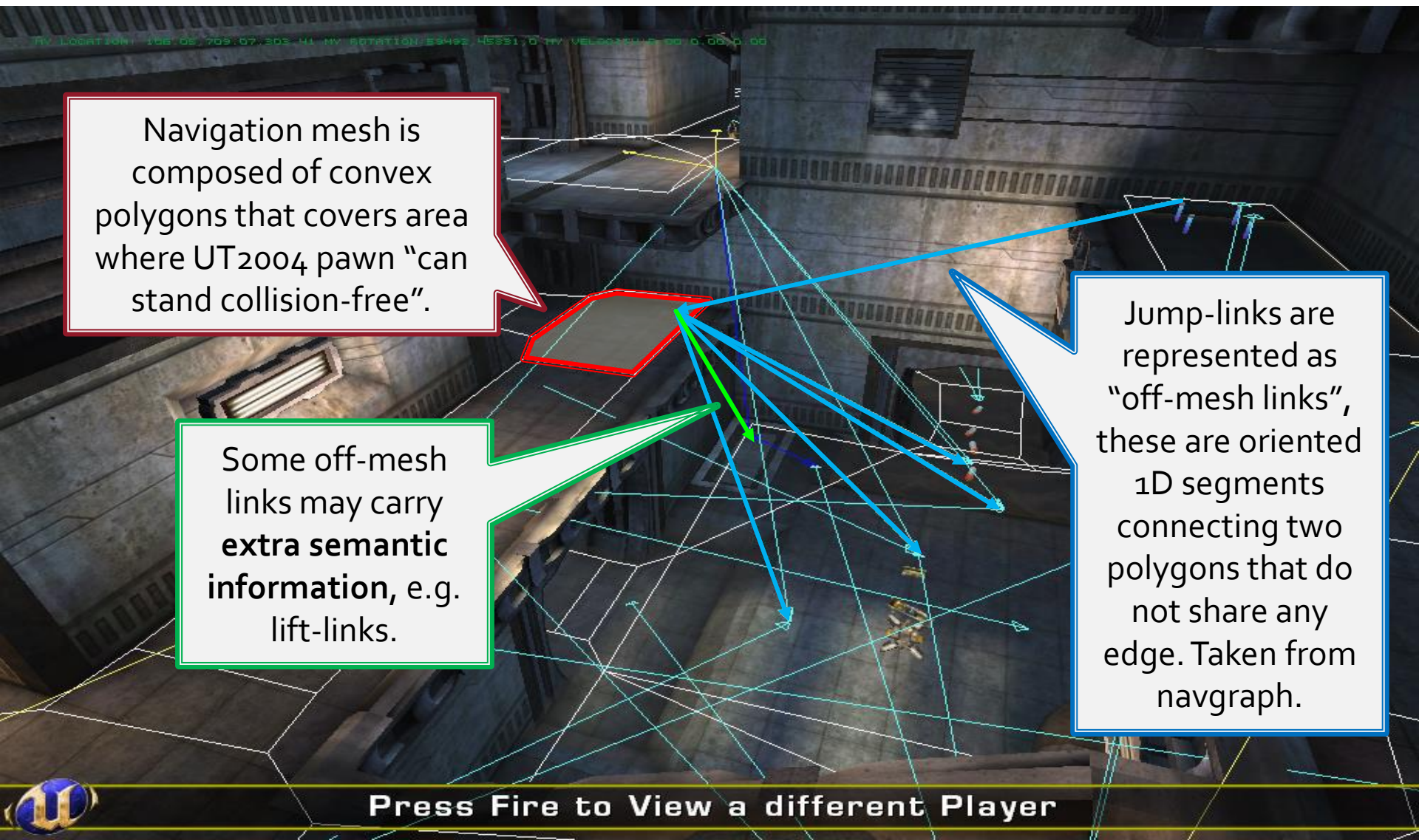
Navigation Mesh

Custom navig. abstr. computed from the map geom.



Navigation Mesh

Custom navig. abstr. computed from the map geom.



Navigation Mesh
Low-level API

Classes of interest:

`NavMeshModule`:

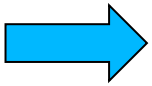
`NavMesh` – „raw“ data vertices, polys, off-mesh points

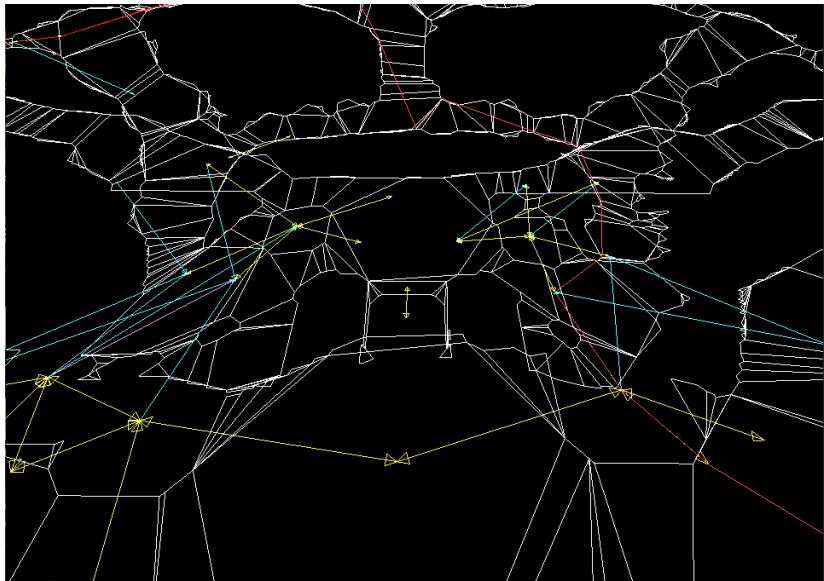
`NavMeshDropGrounder` – find nearest poly

`NavMeshClearanceComputer` – NM “raycast”

Methods of interest:

```
this.navMeshModule
    .getNavMesh()
    .getDropGrounder()
    .getClearenceComputer()
    .getAStarPathPlanner()
```





World Abstraction

`Item, ItemType`

All the small things (details) …

Pogamut World Abstraction

Items overview

Objects - `IWorldObject`

- `Player`
- `Self`
- **`Item`**
- `NavPoint`
- Module for quick sensory readings
  - `this.items`
- Some interesting methods:
  - `items.getItem(UnrealId)`
    - Translation `UnrealId` -> `Item` intsance
  - `items.getItemRespawnTime(Item)`
    - How often is this item respawning
  - `items.isPickable(Item)`
    - Is item pickable given its "spawned" state and my self/inventory state?
  - `items.isPickupSpawned(Item)`
    - Whether the bot thinks the item is spawned within the environment
  - `items.getSpawnedItems()`
    - Map of all items that are believed to be spawned right now

Events - `IWorldEvent`

- `GlobalChat`
- `HearPickup`
- **`ItemPickedUp`**

Pogamut World Abstraction

Items overview

- `Item`
  - More "item spawning location" than item
    - `items.isPickupSpawned(item)`
  - Unique `UnrealId` => Can be used in `Set`, `Map`
  - `ILocated` ~ `getLocation()` ~ X, Y, Z
  - `IViewable` ~ `isVisible()`
  - Usually has corresponding `NavPoint` instance
    - `NavPoint itemNP = item.getNavPoint()`
  - Described by `UT2004ItemType`
    - `item.getType()`

Pogamut World Abstraction

Items overview

- `Item`
  - Not all `Item` instances are "item spawning locations"!
  - `Item.isDropped()`
    - `False` => `Item` describes item spawning location, once picked up, will respawn after some time
    - `True` => `Item` has been dropped by some killed player, once picked up, will never respawn at the same location

Pogamut World Abstraction

Items overview

- `Item.getType()` returns `UT2004ItemType`
  - Enum holding concrete type of the item
  - Part of some `ItemType.Category`
    - Categories are divided based on what items are intended to do
    - `ItemType.Category.HEALTH`
    - `ItemType.Category.ARMOR`
    - `ItemType.Category.SHIELD`
    - `ItemType.Category.WEAPON`
    - `ItemType.Category.AMMO`

"Armor" value a player sees in HUD is a sum of its "armor" and "shield"; different weapons damages/bypasses different "armor types".

Pogamut World Abstraction

Items overview

- `items.getSpawnedItems()`
- Is the item spawned?
  - A crucial question to answer in order to decide what to start navigating to
  - Subject to partial observability of UT2004; if the bot does not see "item spawning point", and it did not pickup the item recently, the bot cannot be sure whether some item is spawned or not
  - The bot can only "believe" that the item is spawned

Pogamut World Abstraction

Items overview

Listening to `ItemPickedUp` event is a bit tricky…

```
@EventListener(eventClass = ItemPickedUp.class)
public void itemPickedUp(ItemPickedUp event) {
   Item item = items.getItem(event.getId());
   …
}
```

Typically, one would think that the callback is raised when you pick some item up ONLY.

However, the problem is that "`ItemPickedUp`" is broadcast also before the bot is spawned into the environment for "initial bot equipment" a bot receives every time it is launched into the environment. Such IDs will not have `Item` instance counterparts!

This is especially problematic for the case when you are spawned into the game for the very first time as you, for instance, have not received `Self` message yet and so, e.g., `this.self` is not properly initialized yet.

Navigation

… and how to decide
where to go?

Navigation

Picking an Item step by step

Navigation means to:

1. Decide where to go
2. Plan the path (list of path points)
3. Follow the path
   - Keep checking that you still want to continue with the navigation.
4. Check that you have truly reached your destination once navigation stops!

Navigation

1. Decide where to go example

```
targetItem =
  DistanceUtils.getNearest(
    items.getSpawnedItems().values(),
    info.getLocation(),
    new DistanceUtils.IGetDistance<Item>() {
      @Override
      public double getDistance(
                        Item object, ILocated target) {
        return
          navMeshModule
            .getAStarPathPlanner()
            .getDistance(target, object);
      }
    });
```

3. Make sure to save the item you're navigating to into your memory.

1. Returns nearest item you think is spawned wrt. your current location.

2. Using path-metric instead of default Euclidean one.

Navigation

2.+3. Plan & Navigate

```
navigation.navigate(targetItem);
if (navigation.isNavigating()) return;
else {
  // item location unreachable from my current
  // position
}
```

- Notice the `else` branch; it might happen that you are trying to start/continue navigation towards unreachable location (depending on your "item selection" routine and your current position)

Navigation

4. Check that you have grabbed the item

```
@EventListener(eventClass = ItemPickedUp.class)
public void itemPickedUp(ItemPickedUp event) {
  if (targetItem != null
      && targetItem.getId() == event.getId())
      itemPicked = true;
}

if (!navigation.isNavigating()) {
  if (itemPicked) {
    // item obtained
  } else {
    if (targetItem “is visible and near”) {
      // move to a location a bit behind the item to grab it
    } else {
      // navigation stuck => you might try to restart it
      // item unreachable => pick different item
    }
  }
}
```

That’s why you need the memory!

Navigation
Cheatsheet

- Deciding where to go
  - `navPoints.getNavPoint()`
  - `DistanceUtils… // Euclidean metric`
  - `fwMap.getNearest(…) // Path-length metric for navigation graph`
  - `navMeshModule.getAStarPathPlanner().getDistance(object, target)`
- Navigation module
  - `this.navigation.navigate(…)`
  - `this.navigation.isNavigating()`
- Stuck listening *(differentiates between "item unreachable" / "stuck")*
  - ```
    this.navigation
            .addStrongNavigationListener(
                    new FlagListener<NavigationState>() { … })
    ```
  - Register this in `beforeFirstLogic()`
- Info about the bot
  - `this.info.getLocation()`
  - `this.info.atLocation(ILocated, epsilon)`
  - `this.info.getNearestNavPoint()`
    ```

Team Communciation

… how to talk to each other?
WhatsApp for bots!

TeamComm Server

Instant messaging for bots

```
public class TeamCommBot extends
    UT2004BotTCController<UT2004Bot>

<dependencies>
    <dependency>
      <groupId>cz.cuni.amis.pogamut.ut2004</groupId>
      <artifactId>ut2004-team-comm</artifactId>
      <version>3.8.1-SNAPSHOT</version>
    </dependency>
</dependencies>
```

Java Example available at:

- svn://artemis.ms.mff.cuni.cz/pogamut/trunk/project/Main/PogamutUT2004Examples/26-TeamCommBot

TeamComm Server

Instant messaging for bots

- TeamComm server is custom solution for sending serializable Java objects (subclasses of `TCMessageData`) between UT2004 bots written in Java
- Requires team-oriented game type to be running (e.g. Team Deathmatch)
- The server understands the game => it makes sure that messages send to the team is not broadcast to members of a different one
- You have to start the TC server manually via:
  - `UT2004TCServer.startTCServer();`
- Once running, you can leave it be (quite stable, handles dis/connection of bots gracefully)

TeamComm Server

Instant messaging for bots

- Module `tcClient`
  - Is connecting to the TeamComm server automatically

```
if (!tcClient.isConnected()) return;
```

  - Provides methods for sending messages to other bots

```
tcClient.sendToTeam(…);
tcClient.sendToTeamOthers(…);
```

- Message is any object that is serializable and extends `TCMessageData`
  - `UnrealId` is serializable
  - `Item`, `Player`, `NavPoint`, … are not!

TeamComm Server

Instant messaging for bots

```
public class TCItemPicked extends TCMessageData {

  private static final long serialVersionUID = 786623423491232L;

  public static final IToken MESSAGE_TYPE =
    Tokens.get("TCItemPicked");

  public UnrealId who;

  public UnrealId what;

  public TCItemPicked(UnrealId who, UnrealId what) {
    super(MESSAGE_TYPE);
    this.who = who;
    this.what = what;
  }

}
```

Java: Each serializable object must has its unique magic number; do not forget to change this when copy-pasting!

TeamComm Server

Instant messaging for bots

```
public class TCItemPicked extends TCMessageData {

  private static final long serialVersionUID = 7866323423491232L;

  public static final IToken MESSAGE_TYPE =
    Tokens.get("TCItemPicked");

  public UnrealId who;

  public UnrealId what;

  public TCItemPicked(UnrealId who, UnrealId what) {
    super(MESSAGE_TYPE);
    this.who = who;
    this.what = what;
  }

}
```

TeamComm: is requiring all messages to define their unique id token; do not forget to change this when copy-pasting!

TeamComm Server

Instant messaging for bots

```
public class TCItemPicked extends TCMessageData {

  private static final long serialVersionUID = 7866323423491232L;

  public static final IToken MESSAGE_TYPE =
    Tokens.get("TCItemPicked");

  public UnrealId who;

  public UnrealId what;

  public TCItemPicked(UnrealId who, UnrealId what) {
    super(MESSAGE_TYPE);
    this.who = who;
    this.what = what;
  }

}
```

TeamComm: is requiring all messages to define their unique id token; pass the token to the super constructor.

TeamComm Server

Instant messaging for bots

```
@EventListener(eventClass=TCMessage.class)
public void allMsg(TCMessage tcMessage) {
    log.info("@EventListener(TCMessage)");
}
```

- Hook for listening to ALL messages TeamComm->Bot
  - Not advised to use for reading, useful for logging while debugging

```
@EventListener(eventClass=TCItemPicked.class)
public void tcItemPicked(TCItemPicked event) {
    log.info("@EventListener(TCItemPicked)");
}
```

- Hook for listening to `TCItemPicked` custom messages
  - Advised way, create separate message handler for every specific message
  - `TCItemPicked` message is user-defined, you may create as many of such a classes as you need

Homework 03

ItemPicker Squad

Revisiting the reasoning

Homework 03
Gotta COLLECT 'em all!

- So now you know how to share information between bots
- How to pick all the items as fast as possible?
  1. One bot will solve the optimization problem and commands the others
     - While you can try to do it … it's resource-demanding => this is not how you would implement in fast-paced games like FPSs; simulation is non-deterministic => pointless to try to plan too much into future
  2. Distributed solution
     - Let it be solved emergently during the simulation itself
     - Each bot is having some information and trying to do the best for itself while cooperating with others
     - You can do a lookahead:
       - Level 0 = be sure not to try to pick something someone else is pursuing
       - Level 1 = be able to interrupt someone else, if you are closer to item other is pursuing
       - Level 2 = assume the common goal and pick the item so the sum of all "current and next run" are the lowest
       - Level 3 = dtto but "current next 2 runs"; usually pointless because of CPU time and simulation non-determinism

Homework 03

Submissions

Submissions will happen through Gdrive again.

Once you finish your homework, ZIP UP your project folder COMPLETELY (except the `target` folder) and upload the ZIP file to shared shared GDrive folder into the `03-ItemPicker` directory.

Questions?

I sense a soul in search of answers…

ASK AT DISCORD!

https://discord.gg/c49DHBJ