Crowds in Hitman:Absolution

Kasper Fauerby

Lead Programmer at IO Interactive

Io-Interactive

GAME DEVELOPERS CONFERENCE
SAN FRANCISCO, CA
MARCH 5-9, 2012
EXPO DATES: MARCH 7-9
2012

Highlevel goals

- Quality over quantity
  - Around 1200 agents per crowd, 500 on-screen
  - Player should not distinguish between crowd & npc actors
- Ambient crowd behaviors
  - Mill around
  - Be aware of points of interest & react to player actions
  - Level designer has partial control of placement & movement flows
- Panic crowd behaviors
  - Evacuate the crowd area
  - Help enhance the action experience of the game
  - Never get in the way of the player during action

Crowds in general

- Two main approaches to crowd simulation:
- Global knowledge / global solution to simulation
  - Continuum based crowds with dynamically updated potential fields
  - Impressive results, especially looking at the actual simulation
  - 10.000 character army charging city gates, near-perfect evacuation of buildings etc.
  - But: Usually requires a limited set of fixed goals
- Agent based
  - No goals (just mill around)
  - Very local behavior
  - Movement can appear very erratic & the individual agents can seem stupid

Crowds in general

- Crowds in a game
  - The ”fun factor” is the most important thing
  - Perfect simulation (no intersections or stopping up) becomes secondary
  - Must be very dynamic and react to player actions
  - Level designers must have quite a lot of manual control
  - Each agent must visually be of an acceptable quality, even when viewed up close
- My opinion
  - The best approach is that of a traditional, but lightweight, AI system


北疆烤羊肉
芝加哥

Our crowd system

- Main components of the crowd system
  - Framework: The cell map, agent model, tools
  - AI: Steering & behavior selection
  - Visuals: animation and character meshes
  - Believability: integration with core gameplay features

The cell map

- We could just add X agents to the world, but:
  - We need very fast navigation mesh queries
  - We need very fast neighborhood queries
  - We need very fast checks for walls & other static obstacles
- We overlay a regular grid on top of the nav mesh
  - This means that the crowd area is only 2.5D (no overlaps in height)
  - Memory usage scales with area of a rectangle, even if walkable region is sparse
  - Each cell stores
    - Walkable/unwalkable flag
    - Current agents in cell (stored as an intrusive singly linked list)
  - Can also annotate the map with additional info, as needed for gameplay

The cell map

- Cell annotations
  - Exclusion zones
  - Panic only cells
  - Ambient flow vectors
  - Teleporters
  - Exit zones

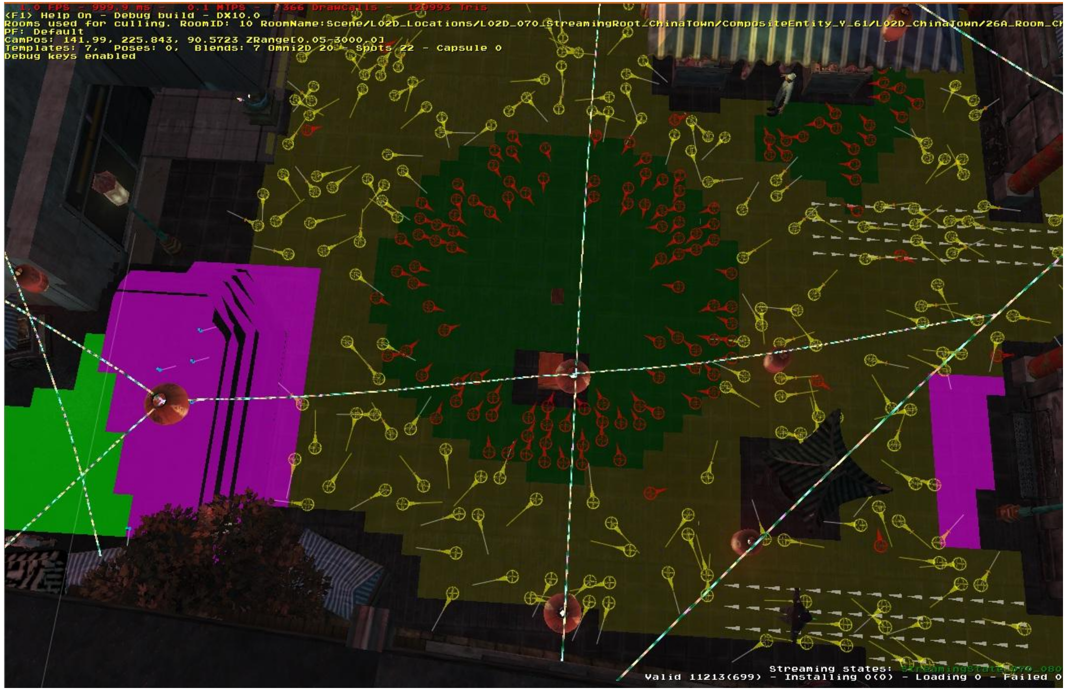
Agent model

- Craig W. Reynolds
- Agent ”particles”
  - Position
  - Radius
  - Forward vector
  - Speed
  - Steering input

Tools: Agent placement

- Agents are distributed onto the cell map as:
  - Manually placed individuals
  - Manually placed groups of agents
  - Randomly placed agents
- Manually placed individuals
  - Originally a debugging tool, but ended up being used quite a lot by level designers
- Groups
  - This is what designers really wanted!
  - Position and shape: spherical or square
  - Agent count
  - Optionally: A list of POIs.
  - Optionally: A list of idle animation overrides

The crowd framework



Crowd AI

- Based on a state machine
  - Basic navigation states: Idle, ”pending walk”, walk
  - Other gameplay states: Alert, dead, possessed, prone, scared etc.
- State specific memory
  - Each state can define a ”memory class” which stores arbitrary AI memory data
  - Placed in fixed-size (small) memory block on each agent
  - Wiped & initialized when entering state
- Every frame the agent ”Thinks”
  - Steps the AI, using current state and current state memory
  - Ask current state if a state change is wanted
  - In some cases, change state randomly

Steering: Pending walk

- Used when
  - Agent is standing still, but wants to be moving.
- Purpose
  - Find the best valid direction and time to start moving
  - Since agent is usually in a crowded place this requires some AI logic
- Sub-phases
  - ”Search for direction”
    - Send out probes to check for wall collisions and other obstructing agents
    - Probe direction is changed every frame, favoring directions in front of agent
  - ”Wait for clear”
    - Wait until agent can start moving
    - Communicate a wanted state change to the agent (into walk state)

Steering: Walk

- Used when
  - Milling about
- Purpose
  - Move agent around, avoiding collisions with walls and other agents
- Algorithm
  - Find preferred direction
    - Check for walls, and steer to avoid collision
    - Check for avoid zones and ambient flows
    - Apply wander behavior (Reynolds)
  - Sample neighborhood for dynamic obstacles, select worst threat (Reynolds)
  - Do ”unaligned collision avoidance” to get actual steering direction (Reynolds+)
  - Either accept the steering, or communitate a wish to stop moving

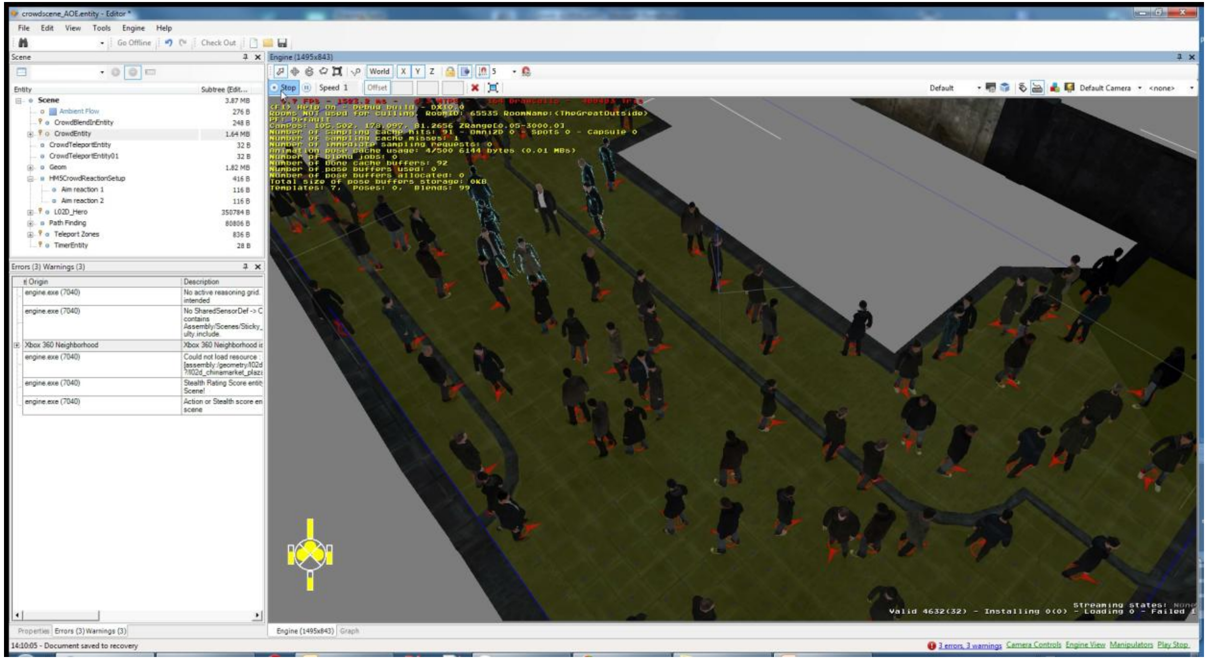
Steering: Panic

- Same as ”Pending walk” / ”Walk”, but tweaked differently
  - Higher speed means different settings for probing for walls, collecting neighbors etc
- Panic steering relies heavily on ”panic flows”
  - Each exit in the crowd becomes one separate ”flow channel”
  - When cell map is generated, each flow channel is calculated
  - Each cell stores: direction to exit, along shortest path, and total cost to reach that exit
  - Each agent dynamically switches between flow channels to quickly flee the map
- Needs some manual guidance/annotation in narrow spaces
  - Panic flows are based on modified Dijkstra algorithm
  - Shortest path generates choke points around corners

Steering: Key learnings

- This turned out to be hard in dense environments!
  - Lots of ”magic numbers” to tweak
  - Especially hard when having multiple movement speeds
- Using speed for steering
  - Turned out to be critical!
  - First decide on a initial *preferred* and *max* speed (for example: *walk relaxed* and *walk fast)*
  - Each steering component (wall or dynamic avoidance) then reports:
    - New *preferred* speed
    - New *maximum possible* speed
  - Decision is based on, for example, distance to wall or whether or not a speed change can resolve a dynamic collision
  - A real human often prefers slightly changing speed over changing direction
- Favor stopping to radically changing direction

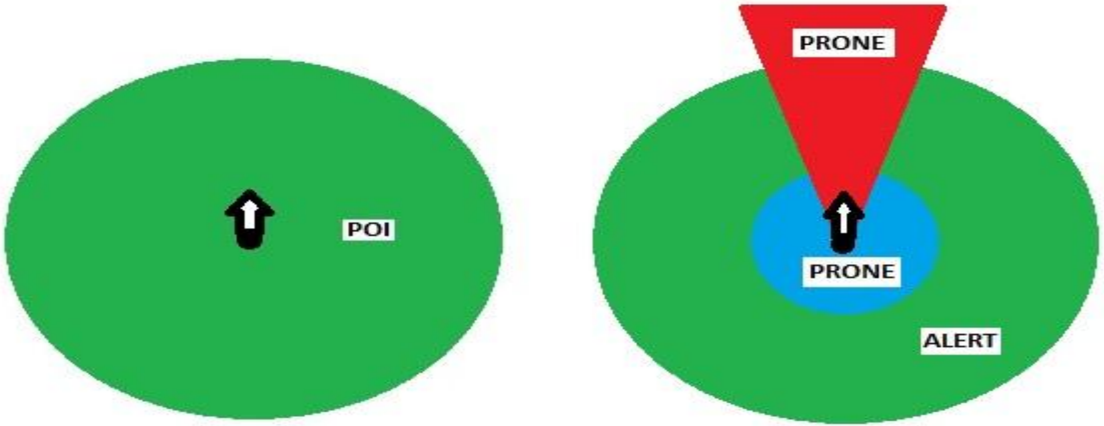
Video: Steering behaviors



Behavior selection

- Navigation AI automatically handles state changes
- More specific AI states are handled differently
  - A data-driven system makes the crowd react to various players actions
  - For example: aiming a gun, shooting, acting suspicious
- A player action spawn up to 3 user-configured zones
  - Radius & angle (spherical or cone)
  - Agent reaction type: (POI, avoid, alert, scare, go prone)
  - Reaction types are listed in ”order of importance”, and a zone can override less important zones

Behavior zones

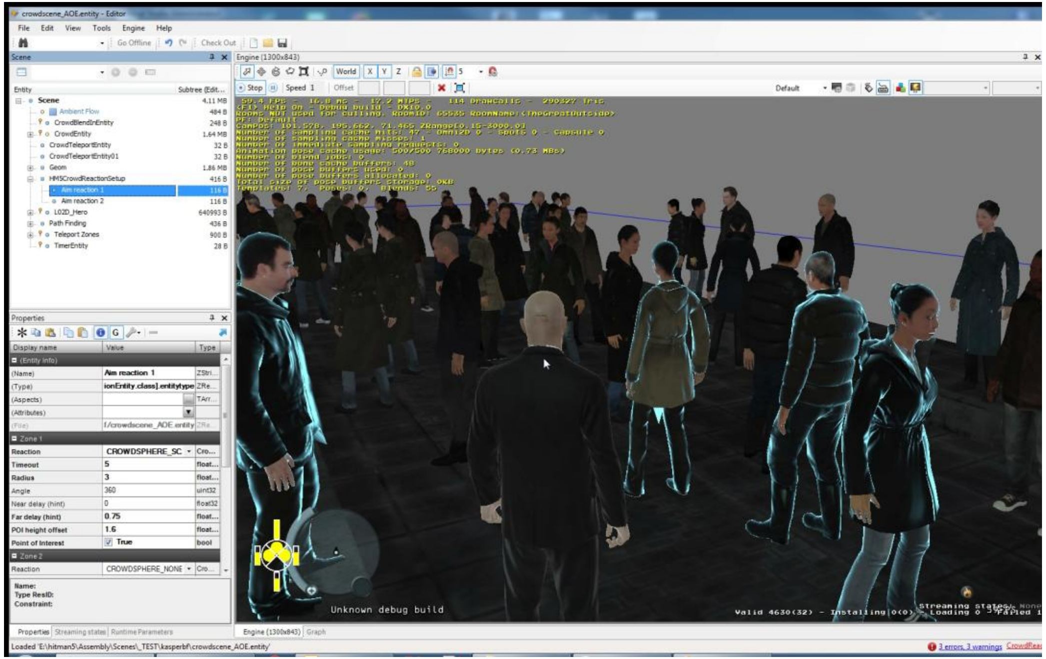


POI
AVOID
ALERT
SCARE
PRONE

Behavior zone pulses

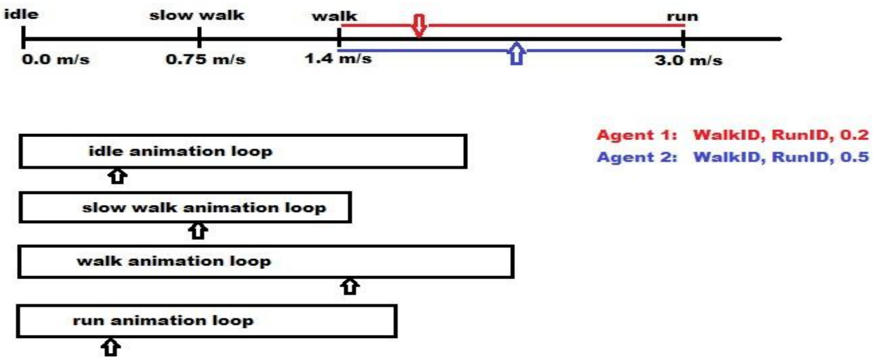
- Zones continously send ”pulses” into the crowd
  - This way each zone ”pushes” itself on the affected agents
- When an agent is hit by a behavior pulse
  - Is this is now the currently most important behavior zone?
    - Check current agent mood (ambient, alerted, scared, paniced, dead)
    - Check ”inflicted mood” from zone (derived from reaction type)
  - During ”Think”
    - If mood for current zone is strictly worse than the current agent mood, then we change AI state
- Benefits of system
  - Level designer configures the behavior on a per-crowd basis
  - Quick and easy way to handle multiple inputs to the agents

Video: Behavior selection



Animation: First attempt

- What and how?
  - Fit animations on top of simulation
  - Share a number of looping clips between all agents (idle, walk, run etc)
  - At any time: animation state for an agent is two animation IDs and a blend weight
- Why?
  - Concerned about animation performance
  - Simple to implement



Animation: First attempt

- Pros
  - Performance was great
  - Navigation logic was stable
  - Agents can move at any speed!
- Cons
  - Overall robotic look and feel
  - Foot sliding in transitions: idle -> walk -> idle
  - No turn/banking animations
  - Agent animation looks synchronized
    - So we added multiple loops per animation, started at random times...
  - Tedious and manual approach to controlling animation state from AI code
  - Code involved in adding new animations to the system
  - Hard to avoid animation glitches and blend errors
- Overall
  - The approach was valid, but we had higher ambitions than that....

Animation: Second try

- What and how?
  - Ambitious goal - 500 agents on screen with no foot sliding plus support for transition and turn/banking animations
  - Based on heavily modified version of ”Near-optimal Character Animation with Continous Control”
    - Annotated motion clips, high-level steering inputs, data driven
  - Agents are now moved by a trajectory channel in the animations, rather than from steering velocity!
  - Each visible agent now needs a uniquely blended animation pose, much like an ordinary NPC
- Why?
  - Player gets very close to the individual agents
  - We felt that having a high quality of animation on each individual was needed for achieving a believable crowd experience
  - Avoid the robotic feel

Animation: Second try

- Pros
  - Looks much better ☺
  - Completely removed tedious animation management code from the crowd AI
  - Greatly simplified the AI code itself
- Cons
  - Took a lot of work to implement and optimize
  - In rare cases a bit more control over the animations can be useful
  - And very importantly: Agents reacts *much* slower to steering input, which makes it harder to avoid collisions and intersections!
- Overall conclusion
  - It was a great success!
  - The approach we used for crowd agents might be how we control real NPCs in future games...

Animation

- Check GDC Vault for: ”**Animation Driven Locomotion for Smoother Navigation”** for further inspiration! (Gabriel Leblanc, Shawn Harris, Bobby Anguelov)

Believability

- Main challenges:
  - Core game mechanics: close combat, human shield etc
  - Detail animation
  - Visual variety

Core game mechanics

- No wish to have duplicate implementation
- Possession system
  - On-demand upgrade agent to full NPC AI
  - Allocates small pool of invisible NPCs
  - Simple API allows game programmers to switch between crowd agent and NPC
  - Made it trivial to support advanced gameplay mechanics

Detail animation

- Head IK
- Crowd acts
  - Talk on phone, smoke, sit on bench
  - Uses possession system and existing cut-scene tools
  - Spawns randomly near player
- Upper body acts
  - Lightweight overlay anims: cough, wave etc.
  - Can play while agent walks around

Visual variety

- Unique scaling factor for each agent
  - Small amount: ~5%
  - Softens up horizon
  - Does wonders for percieved diversity of crowd
- Diffuse texture overrides
  - Simply replace the diffuse texture of material
  - Cheap way of having red shirt, yellow shirt etc..

Performance: PS3

- Some numbers: 1200 agents simulated, 500 on-screen
- PPU: 5ms
  - Animation system: ~2ms
  - Crowd AI / steering: ~2ms
  - Framework: ~1ms
- SPU: ~20ms, distributed across multiple SPUs
  - Animation sampling
  - Animation blending
  - Animation selection logic
  - Frustum and occlusion culling
  - Crowd AI sensors (more later)
- GPU: 8ms
  - Listed here as an example, but obviously very dependent on render tech and meshes used
  - In G2: the vertex shader is limiting factor on PS3 due to skinning massive amount of vertices

Performance

- Scaling?
  - System has very low general overhead
  - Scales nearly linearly with number of agents in crowd
  - Culled/on-screen ratio also affects performance, due to animation cost
- Memory layout: Agent data
  - On the PS3 the memory layout is one of the most important things for performance
  - AI: code is pretty simple, but called many times and:
    - Performs *a lot* of neighborhood searches
    - Inspects properties on all neighbor agents
  - Size of a full agent ~256 bytes
  - Separate out ”agent core”. Stores the most basic properties: position, speed etc. 36 bytes
  - Each agent object has a pointer to its corresponding core
  - Allocate all cores as a single, 128 byte aligned, block of memory (1200 agents: 42kb)
  - Reduces cache missing during simulation and fits on SPUs

Memory layout: Cell map

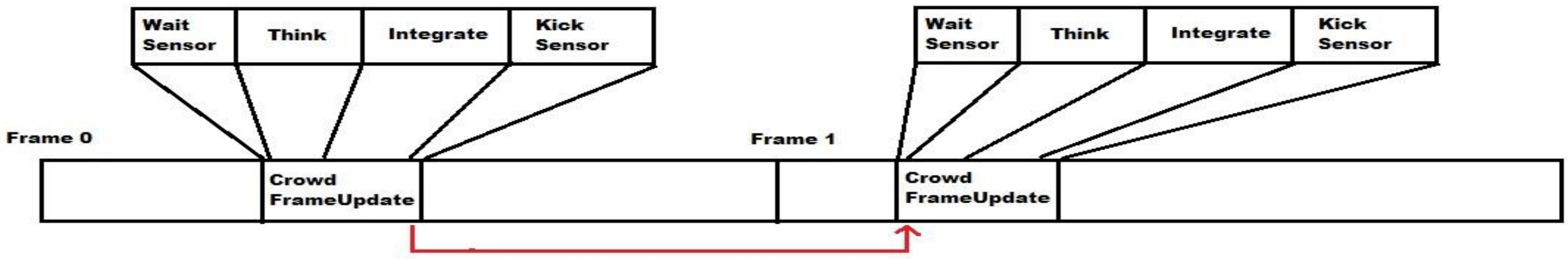
- Conceptually each cell stores many different pieces of data:
  - Walkable/non-walkable (and other "cell flags")
  - Flow vectors
  - Heights
  - Head pointer of linked list of current occupiers
- Bad implementation
  - Implement class ZCell, map is an array of ZCell objects
- Good implementation
  - Map is 4 arrays, each storing a different attribute
- Why?
  - Array of struct vs. struct of arrays
  - Usually an algorithm is only interested in *one* of the attributes
    - Which can then be 128 byte aligned
    - Which can (more easily) fit on SPU local store
    - Spans less memory, in turn causing less cache misses

Crowd AI & steering on SPUs

- Moving the entire AI code to SPU is hard
  - Has many dependencies between components in the system
  - Virtual methods
- Profiling showed a few hotspots
  - Neighborhood gathering
  - Raycasting through cell map
  - Selecting ”worst threat” for steering
  - All hotspots are isolated algorithms, working on a limited input!
- Added sensor system
  - Sensor input: position and radius for neighborhood, raycast requests etc
  - Sensor output: Current neighborhood, current worst threat, ray results

Steering with sensor data

- Sensor input is usually fixed
  - Probe a certain distance ahead of agent for walls
  - Collect around agent
  - Select worst threat
- Sensor input is usually configured once when entering AI state
- Actually, sensor output is not 1 frame delayed
  - (except for first frame in state)



Sensor updates on SPUs

- Each job updates X number of agents
  - So it fans out on multiple SPUs
- Needed data on local storage
  - Agent cores: ~42kb
  - For ray casts: ~16kb
    - Our crowds have around 16k cells
    - Cell flags: Array of bytes
  - For neighborhood searches: ~32kb
    - Head pointers from cell map (stored as 16bit indices)
    - Linked list is intrusive, stored in agent cores
  - Sensor input/output for each of the X agents: ~3kb (30 agents per job)
- In total: ~93kb of data needed. Plenty of room for code.

Conclusions

- We managed to create a new crowd system that is a significant step up from our previous system
- We managed to achieve very good performance, which was necessary since the crowd has to integrate with a full game
- Having a proper layout of data is critical for performance when handling massive amount of characters
- It is a time consuming task to tweak all the magic numbers in steering code
- Having proper animation on characters in very dense crowds is very hard, since steering relies on quick reactions from the characters

Questions?

- (Also feel free to email me at: kasperbf@ioi.dk)

- A big thank you to:
    - Michael Büttner
    - Nis Haller Baggesen
    - Bobby Anguelov