Ubåts Centralen

Projektarbete

Namn:

Anton Gustafsson, Hampus Dahl, Rasmus Johansson, Ahmad Awad

Utbildning:

Python utvecklare inriktning AI

Datum: 2025-09-26

Sammanfattning

Denna rapport beskriver ett projekt kallat Ubåts Centralen, ett system utvecklat i Python för att hantera ubåtsdrönare. Projektet syftar till att implementera objektorienterad programmering för att skapa en modulär applikation som kan hantera ubåtens rörelser, kollisionsdetektering, torped avfyrning med vådaskott kontroll samt bearbetning av sensordata.

Rapporten redogör för systemets funktionalitet, kodstruktur och de tekniska lösningar som har utvecklats. Vidare diskuteras de utmaningar teamet stötte på under projektet och hur dessa hanterades genom agila metoder, versionshantering och kodgranskning. Slutligen lyfts systemets möjligheter till vidareutveckling fram, där den nuvarande arkitekturen ger en stabil grund för framtida expansion och förbättringar.

Förord

Projektet genomfördes som ett grupparbete i kursen Effektiv Python Programmering inom utbildningen Python Utvecklare med inriktning AI. Gruppen vill tacka sina medlemmar för deras insats samt handledare för värdefull vägledning.

1. Projektbeskrivning

1.1 Bakgrund

När det gäller det autonoma systemet, anser gruppens samtliga medlemmar att det är jätteintressant att utforska. I synnerhet komplexiteten dessa system omfattar och användning av objektorienterad programmering (OOP). OOP är ett oumbärligt verktyg för att hantera design, utveckling och underhåll av ett projekt. Det har också en stark egenskap av att hålla koden modulär och lätt att vidareutveckla. Projektet "Ubåts Centralen" syftar till att utforska och tillämpa OOP-principer av ett simuleringssystem för ubåtsdrönare. Bakgrunden är att förstå hur flera autonoma enheter kan koordineras, övervakas och interagera i en dynamisk miljö samtidigt som komplexa scenarier hanteras, som kollisioner och målinriktning.

1.2 Syfte

Syftet med projektet är att utveckla ett funktionellt och modulärt system för ubåtssimulering baserat på goda objektorienterade designprinciper. Systemet ska kunna hantera olika ubåtsoperationer och fungera som en plattform som i framtiden kan byggas ut med mer avancerade simuleringar.

1.3 Mål

Målet med projektet är att utveckla ett centralt system för övervakning och kontroll av Lindas Lustfyllda Försvars ubåtsdrönare. Systemet ska på ett effektivt och tillförlitligt sätt hantera stora datamängder utan att hela filerna laddas in i minnet samtidigt. Projektet syftar till att:

- Implementera en robust objektorienterad arkitektur som möjliggör enkel vidareutveckling, felsökning och testning.

- Utveckla funktionalitet för att läsa in och bearbeta rörelserapporter och sensordata för tusentals drönare, med särskilt fokus på att upptäcka och logga kollisioner.

- Tillhandahålla funktioner för att analysera drönarnas positioner, avfyra torpeder med säkerhet mot vådaskott och hantera aktivering av ubåtars mini-nukes via korrekt krypterad nyckelhantering.

- Skapa ett grafiskt användargränssnitt (GUI) som gör det möjligt för operatörer att visuellt övervaka drönarnas status, sensordata och slutpositioner på ett tydligt och användarvänligt sätt.

- Säkerställa att systemet använder lämpliga algoritmer och datastrukturer (såsom generatorer, lambda-funktioner och dekoratorer) för att optimera både prestanda och läsbarhet i koden.

- Genom att uppnå dessa mål ska projektet resultera i ett skalbart, användarvänligt och tillförlitligt system som stödjer arbetet med att övervaka och styra en stor flotta av ubåtsdrönare.

1.4 Avgränsningar och fokus

Projektet fokuserar mer på kärnlogiken för ubåts simulering och objektorienterad design som inkluderar följande punkter:

- **Grafiskt användargränssnitt (GUI):** Grundläggande implementering med struktur som visar essentiell information istället för en fullt utvecklat grafiskt presentation. Huvuddelen av överflödig information visas i terminalen.

- **Komplex fysik och miljö:** Simuleringen sker i en simpel 2D-miljö där varje ubåt kan röra sig i x och y axeln. Ingen hänsyn behöver tas till komplexa och fysiska modeller, som vattenmotstånd, strömmar eller liknande.

- **Nätverkskommunikation:** Projektet simulerar inte kommunikation mellan ubåtar och de tar inte hänsyn till varandras position i realtid.

- **Avancerad AI för ubåtar:** Ubåtarna använder inte någon form av svärmintelligens då rörelser endast sker genom inläsning av förutbestämd data.

1.5 Tidplan/aktiviteter

Projektet har följt en iterativ utvecklingsprocess och delats upp i typiska sprintar.

- Sprint 1: **Inläsning och hantering av datafiler**
  - Implementering av FileReader-klassen för att läsa in och hantera data från filer.

- Sprint 2: **Grundläggande ubåts hantering och rörelse**
  - Utveckling av Submarine-klassen för att möjliggöra rörelsefunktionalitet.

- Sprint 3: **Kollisionsdetektering**
  - Integration av Collision Checker-klassen med Movement Manager-klassen för att testa olika kollision scenarier.

- Sprint 4: **Torpedo System och vänskap spelskontroll**
  - Implementering av Torpedo System-klassen med logik för att förhindra vådaskott samt utökade testfall.

- Sprint 5: **Sensordata Hantering**
  - Utveckling av SensorManager-klassen och implementering av funktionalitet för kärnkomponenterna, inklusive prestandatester.

- Sprint 6: **GUI-integration och rapportering**
  - Implementering av grundläggande GUI-funktionalitet och förberedelse för projektrapportering och dokumentation.

- Sprint 7: **Applikation eller plattform**
  - Implementering-app med användning PyQt5 i python.

2. Teknisk bakgrund och val

Projektet är utvecklat i Python, ett språk som är lättläst, har ett rikt utbud av bibliotek och erbjuder utmärkt stöd för objektorienterad programmering. Python valdes för att underlätta snabb prototyping och effektiv utveckling.

2.1 Struktur

- FileReader-klass för att läsa data från textfiler.
- Submarine-klass för att spåra enskilda ubåtars positioner och hantera deras rörelser.
- Movement Manager-klass som orkestrerar rörelser för flera ubåtar samtidigt mha generatorer.
- Torpedo System-klass med inbyggd kontroll mot vådaskott.
- Sensor Manager-klass för att bearbeta simulerad sensordata.
- En modulär och testbar kodbas enligt OOP-principer, testad med Unittest, MagicMock och Pytest.
- En grafisk plattform byggd i Python med hjälp av PyQt5.

2.2 Objektorienterad Design

Den objektorienterade designen är ett centralt val för projektet. Genom att modellera verkliga entiteter, såsom ubåtar, rörelsehantering och sensorer, som objekt kunde gruppen uppnå tydlig struktur, modulär kod och enklare underhåll.

- **Inkapsling:** Alla objekt ansvarar för sitt eget tillstånd och beteende, vilket minskar beroenden och gör koden mer robust. Till exempel hanterar Submarine sin egen position och förflyttning internt.

- **Modularitet:** Systemet är organiserat i mappar (core, data, config, utils) som innehåller klasser och logiska metoder, vilket gör det enklare att förstå, testa och underhålla enskilda delar utan att påverka hela systemet.

- **Återanvändbarhet:** Klasser som FileReader och Logger är designade för att kunna återanvändas i andra delar av systemet eller i framtida projekt.

- **Skalbarhet:** Den modulära och objektorienterade strukturen gör det dessutom enkelt att lägga till nya funktioner eller hantera ett större antal ubåtar i framtiden.

2.3 Versionshantering med GitHub

GitHub valdes som versionshanteringssystem på grund av dess robusthet, flexibilitet och status som branschstandard. Plattformen användes för att lagra koden, underlätta samarbete genom pull requests och kod granskningar samt för att hantera projektets utveckling historisk. Detta val gjorde det möjligt för teamet att arbeta effektivt tillsammans.

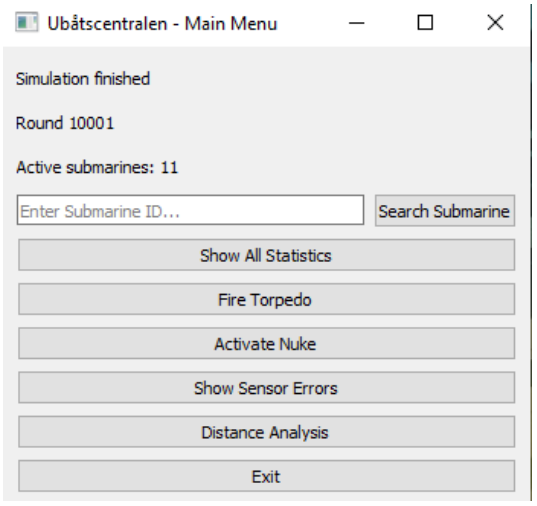
3. Resultat

Projektet Ubåts Centralen gav ett väl fungerande simuleringssystem med resultat som uppfyllde de uppsatta målen. När det gäller implementation för ubåtarnas rörelser så sker den korrekt genom att läsa in rörelsekommandon från filer och för varje runda uppdatera positionerna i den simulerade miljön. Kollisionsdetektering kan identifiera kollisioner mellan ubåtar och markera dem som inaktiva, vilket var konsekvensen av en kollision i denna simulation.

Klassen Torpedo System ger respons om potentiella risker för vådabeskjutning av ubåtar. När ubåtarna nått sin slutdestination kollar klassen om enskilda ubåtar har fritt läge att avfyra i riktningarna upp, ner och fram utan att träffa en annan ubåt. När mini-nukes aktiveras hämtas en nyckel- och aktiveringskod som hashas tillsammans med dagens datum. I simulationen behöver användaren inte ange någon kod för aktiveringen då detta sker automatiskt.

Inläsning av sensordata för att identifiera fel och mönster ger insikter om sensorernas operativa status. Informationen om allt detta lagras i separata filer för enkel avläsning av det som sker i klasserna.

Kodbasen ger ett tydligt strukturerat projekt som är lätt att navigera och förstå, vilket underlättar framtida underhåll och vidareutveckling.



Exempel på Utdata från Simuleringen.

```
Sub 99498042-56 moved forward 6 → pos (26304, 1240)
```

4. Teamarbete, Svårigheter och Lärdomar

Gruppens samarbete har präglats av stabilitet och kontinuitet från projektets inledning. Genom tydlig kommunikation och gemensamt problemlösande har gruppen successivt kunnat hantera de utmaningar som uppstått, vilket i sin tur har bidragit till en fördjupad förståelse både inom programmering och i utvecklingen av sociala färdigheter i en utvecklingsmiljö. De mest komplexa momenten har bestått i att tolka problemställningar, implementera relevanta datastrukturer och designmönster samt att integrera projektets olika delar till en sammanhängande helhet.

En central framgångsfaktor har varit att gruppen tidigt etablerade en tydlig och välorganiserad objektorienterad struktur. Denna struktur har inte bara gjort vidareutveckling och felsökning mer överskådlig, utan även minskat risken för fel och sparat tid i utvecklingsarbetet. Jämfört med gruppmedlemmars föregående projekt har arbetet med att utforma denna struktur varit betydligt enklare, eftersom medlemmarna redan då hade tränat på att tillämpa objektorienterade principer. Den erfarenheten gav gruppen en stabil grund att stå på, vilket möjliggjorde att medlemmarna kunde rikta större fokus på implementation, effektiv integration och optimering.

Utöver detta har gruppmedlemmarna erhållit nya insikter om hur språkspecifika tekniker som generatorer, dekoratorer, lambda funktioner och property påverkar både programmets prestanda och dess läsbarhet. Genom att använda dessa konstruktioner på ett medvetet sätt har gruppen inte bara förbättrat effektiviteten i kodens exekvering, utan även skapat en mer lättförståelig och underhållsbar kodbas. Dessa erfarenheter har varit särskilt värdefulla då de stärker gruppmedlemmarnas förmåga att skriva kod som är både robust och anpassningsbar för framtida utveckling.

En bra lärdom gruppmedlemmarna tar med sig från projektet är vikten av att säkerställa med produktägaren att det som utvecklas, är det som efterfrågats. Detta genom återkommande avstämningar för att undvika onödigt arbete.

4.1 Hur Medlemmarna Arbetade Tillsammans

Från projektets planering har gruppen använt en väldefinierad arbetsstruktur samt anammat agila principer och metoder för att hålla processen flexibel och iterativ. Externa verktyg har varit ett bra sätt att planera gruppens och individuella medlemmars uppgifter. Det gav en god översikt av vad som behöver göras, vad som gjorts och projektet kunde tydligt byggas steg för steg.

- **Initial Planering och Design:** Gruppen bestämde sig för en modulär design där varje kärnfunktion (rörelse, kollisioner, sensorer) skulle isoleras i sin egen klass. Detta gjorde arbetet parallellt utan att störa varandras kod i onödan.

- **Ansvarsfördelning:** Varje medlem tog ägandeskap över en eller flera komponenter. Exempelvis tog en medlem Movement-Management klassen, en annan tog Torpedsystem klassen och så vidare när det gäller andra klasser som datahantering.

- **Kommunikation:** Gruppen bestämde sig för att använda Teams verktyget för direkt kommunikation i skrift eller virtuella möten. Kommunikationen innefattade ofta frågor och dagliga uppdateringar. Gruppen höll även veckovisa möten där medlemmar demonstrerade sina framsteg, diskuterade hinder och planerade nästa steg. Detta gjorde projektets övergripande status tydlig samt att säkerställa att projektet fortskred som planerat..

- **Versionshantering med Github:** Verktyget var ryggraden i samarbetet för att dela kodbasen. “Branches” användes för att testa nya funktioner eller större ändringar. När en funktion skapades och testades så skapades en “pull request” (PR). Denna process blir som en kodgranskning för att upprätthålla hög kodkvalitet och fånga buggar tidigt.

4.2 Svårigheter

Under projektets gång stötte gruppen på flera specifika utmaningar:

- **Logiska buggar i kollision detekteringen:** Den första implementationen av kollisionshanteringen visade sig innehålla logiska brister. Detta ledde till ett tidskrävande felsökningsarbete och fördröjningar i projektplanen. Arbetet gav gruppen dock en djupare förståelse för hur generatorer fungerar och hur de kan användas för att effektivisera kodflöden. Även om det innebar ett initialt slöseri med tid, bidrog processen till en värdefull lärdom om vikten av noggrann testning och stegvis verifiering.

- **Merge-konflikter i Git:** Versionshanteringen orsakade återkommande problem, framför allt vid hantering av push/pull-rutiner, merge-konflikter och konfliktlösning. Trots tidigare erfarenheter var procedurerna stundtals förvirrande, vilket ledde till misstag och ytterligare tidsåtgång. Dessa svårigheter påminde gruppen om vikten av gemensamma riktlinjer för Git-arbete och behovet av tydligare rutiner för samarbete i kodbasen.vi

- **Implementation av teori i praktiken:** Lektionerna gav gruppen de teoretiska kunskaper som behövdes för projektet, men i praktiken visade det sig ofta vara mer utmanande att omsätta dessa i fungerande kod. För att lösa detta fick gruppmedlemmarna aktivt söka information, analysera dokumentation och jämföra olika lösningsstrategier. Denna process var stundtals krävande, men den hjälpte gruppmedlemmarna att både fördjupa sin förståelse och utveckla sin förmåga att själva hitta och tillämpa ny kunskap.

5. Avslutning och slutsatser

Projektet "Ubåts Centralen" är ett bra exempel på objektorienterad programmering i Python, det hanterar olika komplexa simuleringsuppgifter som ubåtsrörelser, kollisioner och sensoravläsning. Det hanterar de utmaningar projektet medför på ett effektivt sätt genom att applicera objektorienterad programmering. Gruppen arbetar agilt inom utveckling, versionshantering, kodgranskning och kontinuerlig testing. Den objektorienterade designen bidrar till projektets underhållbarhet och framtida utbyggbarhet. Det utgör en solid grund för vidare utveckling och utökning av funktionalitet inom ubåtssimulering och kontrollsystem.

6. Referenslista

https://docs.python.org/3/c-api/gen.html

https://docs.python.org/3/library/functions.html

Kursen: Effektiv Pythonprogrammerings föreläsningar.