RoboHockey

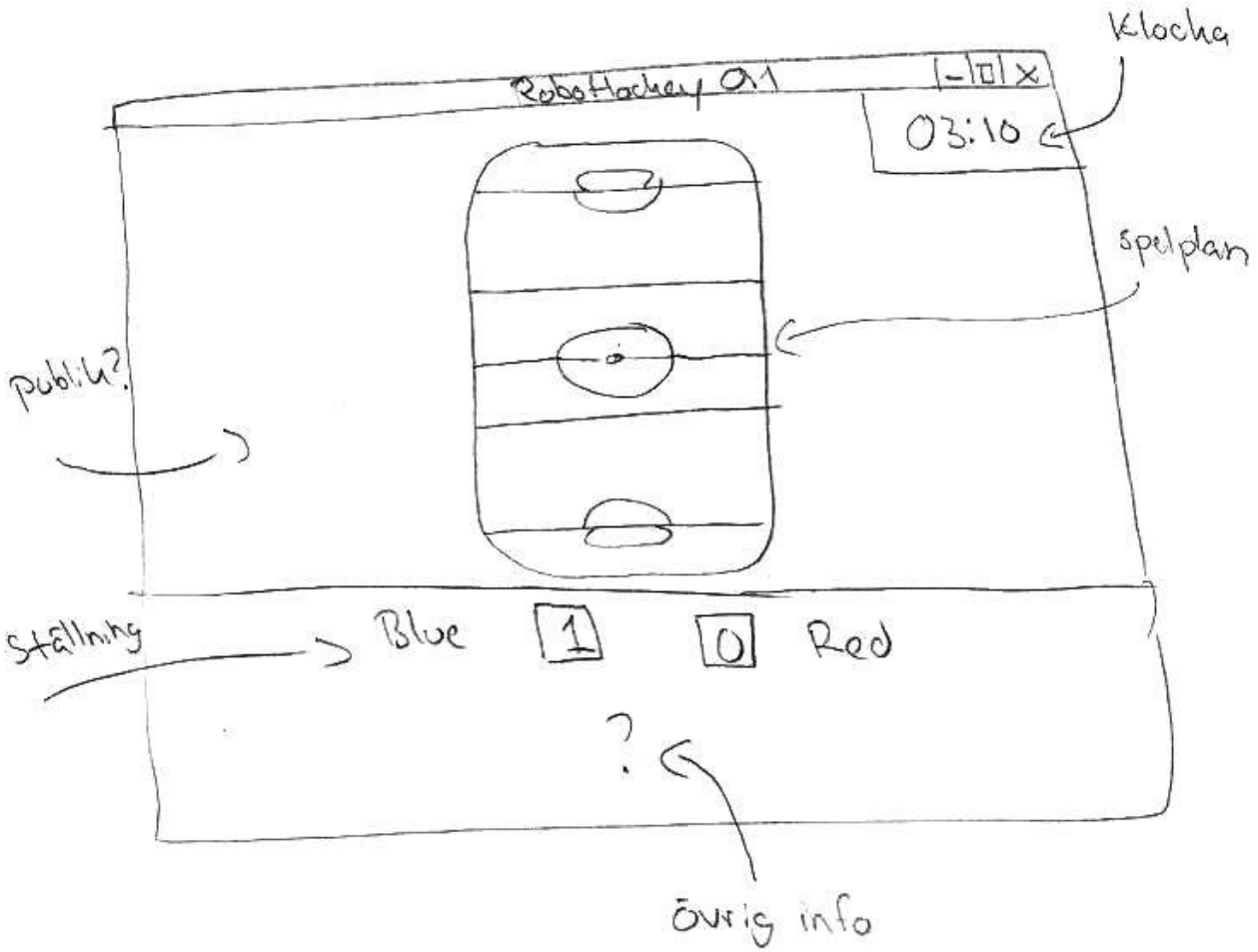
Inledning

Robohockey ska vara ett program som simulerar en hockeymatch 1 mot 1 där datorn spelar mot sig själv. Programmet ska skrivas i C++ med hjälp av biblioteket SDL (http://libsdl.org). Versioner av biblioteket finns för alla tillgängliga plattformar.

Programmet ska bestå av ett grafiskt gränssnitt som beskriver en hockeyplan i 2D med spelarna representerade som olikfärgade cirklar. Pucken representeras som en svart cirkel. I mån av tid kan även klockan och ställningen visas på skärmen.

Programmet ska köra i realtid, dock finns inga krav på att hastigheter ska stämma utan detta görs på en höft för ett gott visuellt resultat. Dock ska hastigheterna vara samma även om frameraten varierar.

Skissen nedan visar gränssnittets schematiska utseende.



AI

AI:n i spelet består i att spelarna ska agera på ett korrekt sätt beroende på om de har pucken eller inte. Vid puckinnehav ska spelaren gå till attack mot motståndarens mål. Den andra spelaren ska då försöka försvara målet och ta pucken av spelaren med pucken. Spelaren med pucken ska även i

varje läge utvärdera om det kan vara värt att ta ett skott mot mål. Planen skall då vara indelad i zoner och sannolikheten för att göra mål med ett skott ökar i zoner med högre sannolikhet. Till exempel är sannolikheten högre att ett skott resulterar i mål om spelaren befinner sig nära motståndarens mål.

Om spelaren utan puck kommer tillräckligt nära spelaren med puck skall denne utföra en tackling vilket resulterar i att pucken byter förare. Om spelaren utan puck dock inte befinner sig tillräckligt nära skall denne försöka tvinga puckföraren till ett så dåligt skottläge som möjligt.

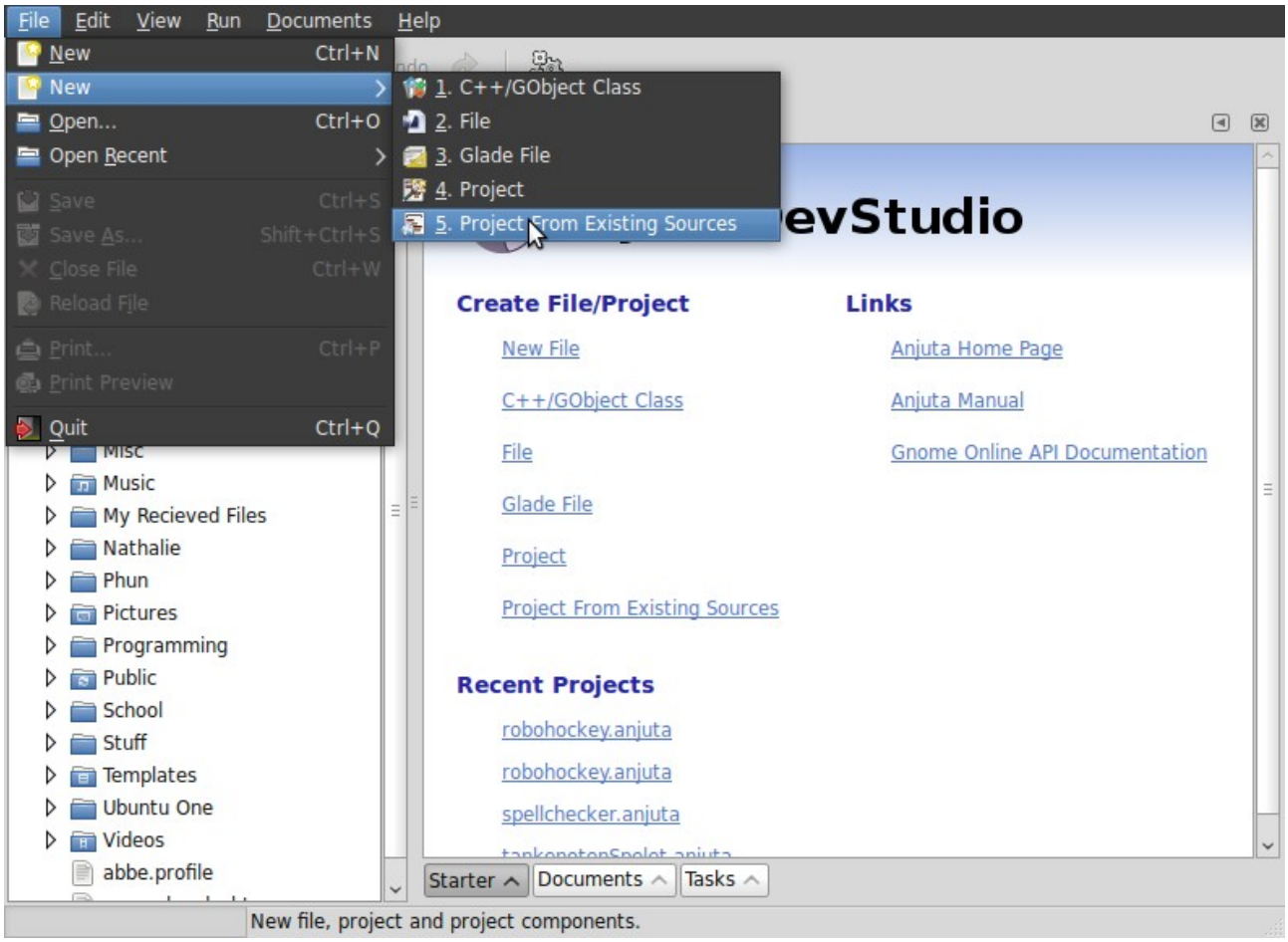
Praktiska detaljer

Källkoden till projektet kommer att versionshanteras med git (http://git-scm.com/) och hostas på github.com, förmodligen. Som byggsystem kan GNU autotools eller valfri utvecklingsmiljö användas. För att använda utvecklingsmiljön Anjuta i Linux kan följande göras:

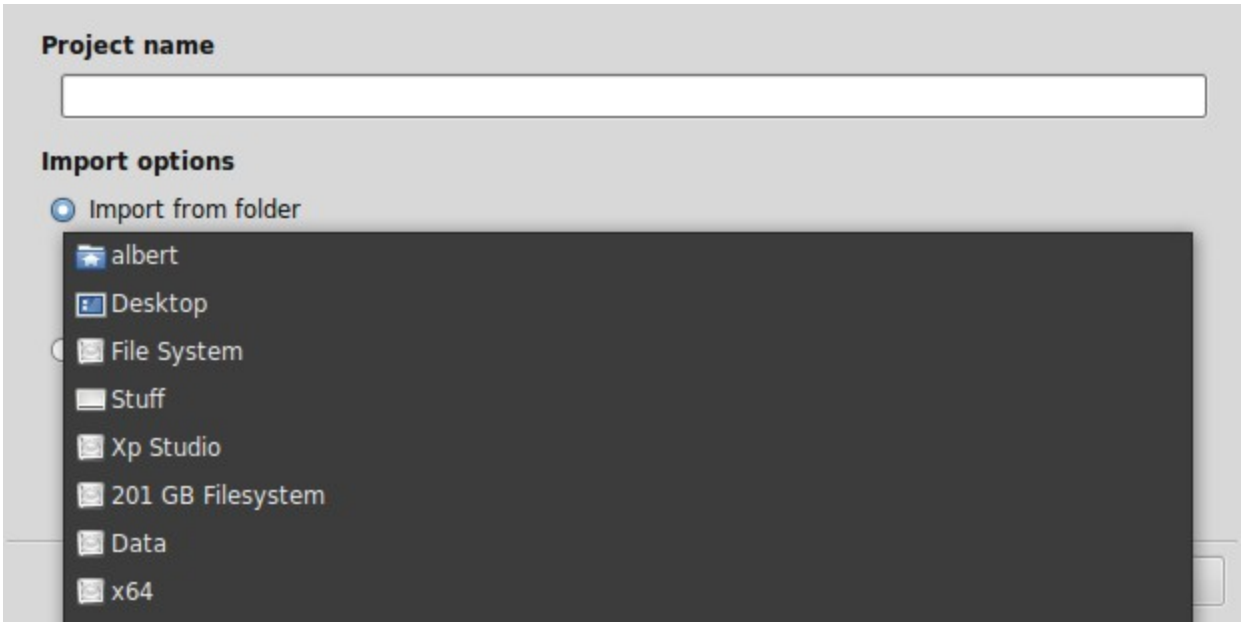
1. Se till att källkoden finns på din dator (via git).

2. Öppna Anjuta

3. Välj *File->New->Project From Existing Sources* (andra namn på svenska)



4. Fönstret nedan kommer då upp. Välj *Import from folder* och sedan väljer du mapp. Om den inte finns i listan tryck bara på *Other...* så visas en filbläddrare där du letar reda på projektets mapp (toppnivån, inte inne i mappen src eller så).

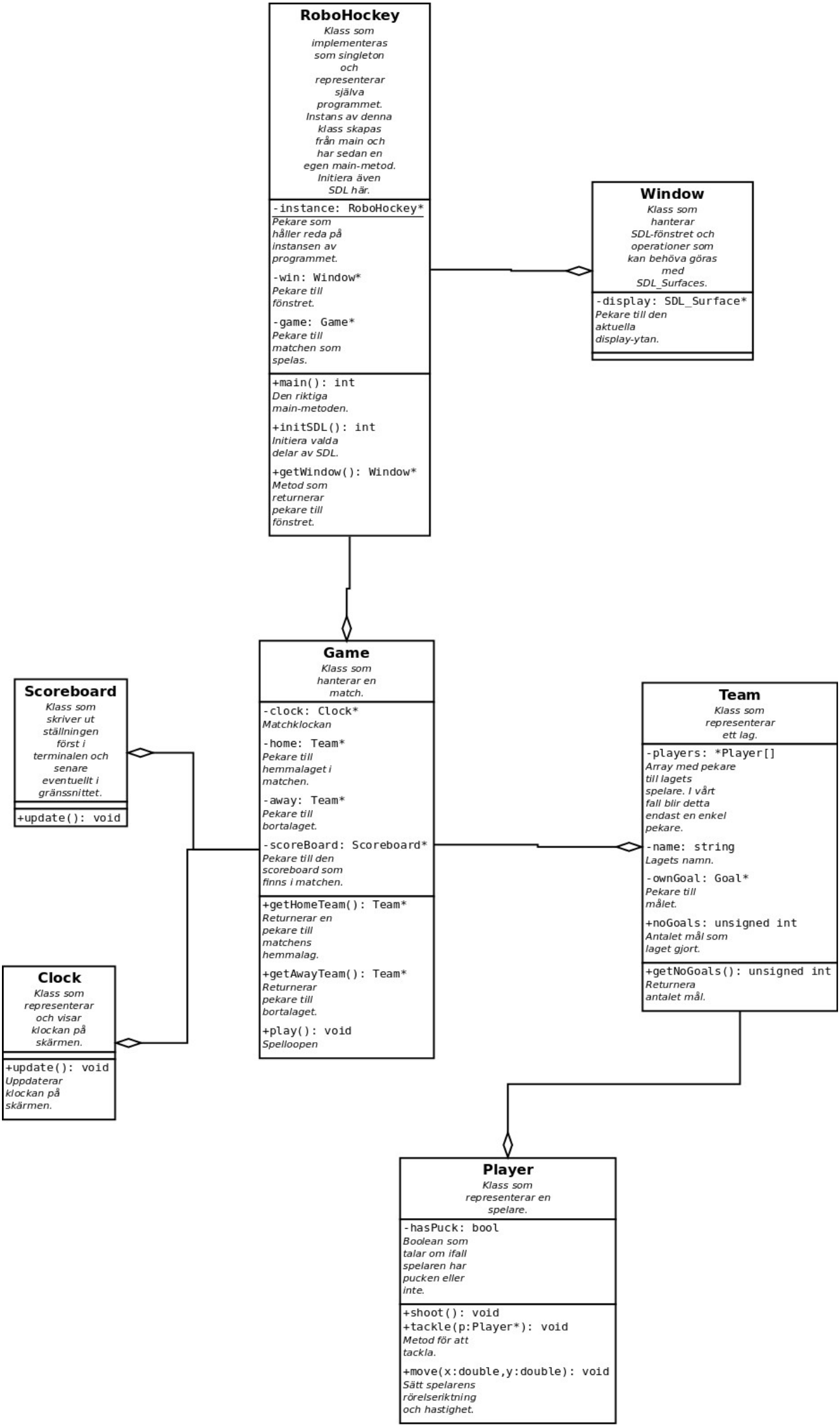


5. När mappen har valts kan du fylla i ett vettigare projektnamn om inte det som står där duger. Sen är det bara att gå vidare och köra på!

Implementering

Då projektet skall utnyttja SDL föreslås att en liten koll tas på SDLs dokumentation.

Nedan följer ett UML-diagram med tanken för implementeringen. Detta diagram är endast en enkel skiss och ska inte ses som någon sanning, endast en idé. Det är dock en idé om vad som ska implementeras.


RoboHockey
Klass som implementeras som singleton och representerar själva programmet. Instans av denna klass skapas från main och har sedan en egen main-metod. Initiera även SDL här.
-instance: RoboHockey*
Pekare som håller reda på instansen av programmet.
-win: Window*
Pekare till fönstret.
-game: Game*
Pekare till matchen som spelas.
+main(): int
Den riktiga main-metoden.
+initSDL(): int
Initiera valda delar av SDL.
+getWindow(): Window*
Metod som returnerar pekare till fönstret.
Window
Klass som hanterar SDL-fönstret och operationer som kan behöva göras med SDL_Surfaces.
-display: SDL_Surface*
Pekare till den aktuella display-ytan.
Game
Klass som hanterar en match.
-clock: Clock*
Matchklockan
-home: Team*
Pekare till hemmalaget i matchen.
-away: Team*
Pekare till bortalaget.
-scoreBoard: Scoreboard*
Pekare till den scoreboard som finns i matchen.
+getHomeTeam(): Team*
Returnerar en pekare till matchens hemmalag.
+getAwayTeam(): Team*
Returnerar pekare till bortalaget.
+play(): void
Spelloopen
Scoreboard
Klass som skriver ut ställningen först i terminalen och senare eventuellt i gränssnittet.
+update(): void
Team
Klass som representerar ett lag.
-players: *Player[]
Array med pekare till lagets spelare. I vårt fall blir detta endast en enkel pekare.
-name: string
Lagets namn.
-ownGoal: Goal*
Pekare till målet.
+noGoals: unsigned int
Antalet mål som laget gjort.
+getNoGoals(): unsigned int
Returnera antalet mål.
Clock
Klass som representerar och visar klockan på skärmen.
+update(): void
Uppdaterar klockan på skärmen.
Player
Klass som representerar en spelare.
-hasPuck: bool
Boolean som talar om ifall spelaren har pucken eller inte.
+shoot(): void
+tackle(p:Player*): void
Metod för att tackla.
+move(x:double,y:double): void
Sätt spelarens rörelseriktning och hastighet.

Tidsplanering

v 16: Påbörja implementering.

V 17: Det visuella ska vara klart. Hockeyplanen och spelarna ska alltså synas och vara möjliga att flytta på. Klocka och ställning kan sparas till senare och ställningen kan med fördel skrivas ut i terminalen så länge.

V 18: Implementering av AI skall påbörjas. Att börja med kan vara att se till att spelarna åker åt rätt håll.

V 19: Implementering av AI skall vara klar. En match ska alltså kunna spelas. Extra småsaker kan göras här.

V 20: Testning och småsaker. Redovisning av projektet på torsdag.