1 Diepgaande Beschrijving van het project

1.1 Diepgaande uitleg

In deze sectie zal onze interpretatie van de opgave worden uitgelegd. Deze interpretatie is voorgelegd aan de opdrachtgever en is goedgekeurd. De volledige visuele voorstelling van de applicatie wordt getoond in de Sectie **??** Mockups van dit analyse verslag.

Zoals al eerder vermeldt zal een programma zijn flow kunnen opbouwen in het deel van de IDE dat we het Wired-view noemen. Hierin zullen instanties van klassen die eerder gedefineerd werden de mogelijkheid worden geboden om informatie aan elkaar door te geven. Deze informatie noemen we een Event. Wanneer een instantie een Event verstuurd en wat hij met een ontvangen Event doet is beschreven in de klasse waartoe hij behoort.

In de volgende paragrafen zal een diepgaande uitleg worden geven over wat een Events is en hoe de gebruiker er gebruik van maakt alsook hoe hij een klasse kan definiëren en welke standaard eigenschappen een klasse in de IDE bevat.

1.1.1 Events

Wat stelt een Event voor?
Om onderlingen communicatie tussen Instanties van Klassen voor te stellen, gebruiken we een Event. Een Event kan al dan niet informatie bevatten. Een Event zonder informatie kan beschouwd worden als een trigger. De informatie dat een Event kan bevatten kan uit meerdere delen bestaan. De infomatie kan dus bestaan uit meerdere primitieve types (int, string of boolean). Elk deeltje in die informatie noemen we een variable. Met elke variable wordt een naam geassocieerd. Deze kan de gebruiker dan gebruiken om de variabele uit een Event op te vragen. Ook het Event zelf moet een ID hebben dat als type geldt.

Eens de gebruiker een Event heeft gedefineerd in de daarvoor voorziene omgeving kan hij er verder in de IDE gebruik van maken. Dit doet hij dan door er een EventInstance van aan te maken. Hij kan dus vanaf dan een Event het eerder gedefineerde type aanmaken en invullen met de informatie die hij wenst mee te geven. Het zenden van een EventInstance door een klasse noemt een emit. Naar welke instanties van klassen het EventInstance wordt verzonden, kan worden bepaald in het Wired-view van de IDE.

Visuele voorstelling: Wired-view van het canvas.
Er is een aparte view waarin alle Instanties van Klassen als blokjes getoond worden. Deze blokjes bevatten inkomende en uitgaande poorten. Deze stellen respectievelijk de evenementen voor die een Instantie wil ontvangen en de evenementen die het uitzendt. Er kunnen verbindingen gemaakt worden tussen de

uitgaande poorten van een instantie en de inkomende poorten van een andere instantie.

Dit aparte view is echter de begin positie van alle gewenste Instanties van de aangemaakte classes. De gebruiker heeft de optie om de verbindingen al dan niet te tonen. Voor de debug-modus zou dit view statisch zijn. Terwijl een extra view het eventueel bewegen van de instanties toont. De gebruiker kan zo de flow van Events bekijken.

Creatie Events.
Nieuwe Events kunnen aangemaakt worden door de gebruiker in een aparte sectie van de IDE. Een Event moet een type hebben, vervolgens kan er informatie meegegeven worden aan dit Event. Deze informatie is een POD (plain old data) die opgebouwd wordt door de gebruiker. Hierin zal elke variable een unieke naam en specifiek type hebben. Het doorgeven van Events door/aan specifieke instanties werd beschreven in **??**.

Standaard events.
Er zijn standaard events beschikbaar zoals oa. onKeyPress, onClicked, onStart, enz. Deze events zijn voorgedefineerd en dienen om interactie te hebben met het visuele canvas.

1.1.2 Klassen

Een Klass kan worden vergeleken met een Sprite in de visuele programmeeromgeving Scratch [**?**]. Het verschil in deze applicatie is dat de instanties expliciet aangemaakt worden in de wired-view. Een Klasse bestaat uit: input Events, Handlers voor die Events, functie definities en member variabelen. Een Klasse kan worden voorgesteld in het Wired-view. Deze appearance kan door een functie in de Klasse worden veranderd. Een Instantie kan dan in het Wired-view beslissen van welke andere Instanties het die input Events ontvangt of naar welke instaties hij Events verstuurd.

Een Klasse kan op **Events ontvangen**. Dit werd besproken **??**. Het afhandelen van een Event gebeurt door een handler die het event binnen krijgt. Het raadplegen van de inhoud van een Event, kan doormiddel van een accessblok.

Een Klasse kan **Events emitten**. Dit kan met behulp van een Emit blok. Hierin moet een Event worden geplaatst. Als een Event informatie bevat zal deze ook hier moeten worden ingevuld.

Een Klasse zal ook een overzicht hebben met alle Events die erdoor worden geemit.

Een uitbreiding van visuele omgeving door toe te laten om **functie aanroepen** te maken binnen een Klasse. Eerst was het idee om dit voor te stellen met een lijn die twee functieblokken zou verbinden. Bij een Klasse met veel interne functie aanroepen wordt dit echter onoverzichtelijk.

Figuur **??** is een voorbeeld van een interne aanroep. Een functie oproep van uit een andere functie. Dit zal worden voorgesteld door een pijl naar een ander, kleine blokje. Dit blokje bevat de naam van de functie die zal worden opgeroepen. Alsook zijn input parameters. Hierin kunnen variable gebruikt worden die als constante worden doorgegeven. De onderkant van een functieaanroep blok bevat een leeg vakje voor de return waarde. Hier kan een variabele aan gekoppeld worden om deze waarde op te vangen.

Member Variabelen zijn variabele die gelden per Instantie van een Klasse. Deze kunnen bijvoorbeeld de positie van de Instantie van de Klasse in het canvas voorstellen.

Visuele voorstelling van een Class.

Er wordt een virtueel canvas en een console geïmplementeerd. Een instantie van een Class kan dus al dan niet voorgesteld worden door een afbeelding.

[scale=0.20]"../voorstel project/mockups/eventcreatie".jpg

Figure 1: Creatie van events.

[scale=0.2]"../voorstel project/mockups/functiecalls".jpg Functiecalls.

1.2 Blokken

Een blok is een blokje dat de gebruiker kan plaatsen in het programmeer venster van de IDE. Deze blokken kunnen alles voorstellen, bv. variabelen, types, control-flow, functions, enz. .

Elke blok die gezet kan worden in control-flow blokken of functie/handler blokken zijn primitieve steps. De expressie (1 + (2 + 2)) (hierbij duiden de () op een block, in dit geval een arithmetic operator block). Deze expressie telt als één primitieve step.

1.2.1 Een proces

Een proces is een gesimuleerde thread. Deze beheert nodige data zoals een variable-stack en code. Een proces wordt uitgevoerd door de VM. Een proces kan gerunned worden en deze zal dan één primitieve stap **??**uitvoeren.

1.2.2 Een variable

Een variable is data die gebruikt wordt door het programma dat gemaakt is. Dit heeft een bepaald type zoals een number of string. Er kan geschreven worden naar een variable en de variable kan gelezen worden.

1.2.3 De Virtual Machine

De virtual machine is hetgeen dat de processen beheerd en uitvoerd. Deze zal zorgen voor het concurrent uitvoeren van de verschillende processen. Voor events te gebruiken maakt hij gebruik van een EventDispatcher **??**.

1.3 Prioritair functies

Als hoogste prioriteit hebben we een stripped down versie van de "programmeertaal" gekozen. Dit zodanig dat we een simpele versie hebben om programma's te kunnen testen. Deze stripped down versie zal de meeste features van de programmeertaal implementeren. Niet alle blocks zullen dan geïmplementeerd worden, maar de VM, proces etc zullen afgemaakt zijn.

Hierna bouwen we hierop de IDE en de XML parser. Er zal dan een ruw prototype aanwezig zijn van de Visuele IDE. De programmeertaal zal dan verder afgemaakt worden, alsook de professionele look van de GUI. Als er hierna nog tijd is, kunnen extra's geïmplementeerd worden.

1.4 Extra's

Er zijn enkele features die we als extra gelaten hebben.

1.4.1 Data types

Een eerste extra zijn lists of meer in het algemeen, extra primitieve data types. Momenteel zijn er maar enkele primitieve types gepland. Namelijk: numbers, string, booleans, events en literal values. Er is geen mogelijkheid om lijsten te kunnen aanmaken, alsook geen mogelijkheid voor integers, etc. Extra types zijn altijd welkom in een programmeertaal. In het design dat we gemaakt hebben kan op simpele wijze nieuwe types aangemaakt worden.

1.4.2 Static functions

Static functions zijn ook gepland als extra. Dit zijn functions die opgeroepen kunnen worden door alle Classes en dus niet behoren tot een bepaalde Class. Een voorbeeld gebruik zou kunnen zijn: stel dat de gebruiker een derde-machts wortel functie wilt maken. Deze functie moet momenteel behoren tot een bepaalde Class. Dus deze functie moet ofwel opgeroepen worden via een soort event, of via code-duplicatie de functie in elke Class implementeren. Een static functie zou dit probleem oplossen. Dit kan ook simpel toegevoegd worden via ons design.

1.4.3 Multiple returns

De mogelijkheid voor meerdere return values bij een functie is een extra. Onze "programmeertaal" biedt de mogelijkheid tot meerdere return waardes, echter zal in de GUI dit niet geïmplementeerd worden. Dit implementeren in de GUI zien we als extra.

1.4.4 Instanties op runtime aanmaken

Momenteel kunnen instanties enkel op creation time aangemaakt worden. Tijdens het runnen van een programma kunnen er niet dynamisch extra instanties aangemaakt worden. Dit is toch een interessante extra, aangezien dit een heel krachtig mechanisme is. In ons huidig design kan er gemakkelijk een clone functie (om instanties om runtime aan te maken) toegevoegd worden.

1.4.5 Collision detection

Functions om collision detection te testen tussen verschillende instanties zien hebben we als extra genomen.