URWPGSim2D Developer Guide

V 1.2 Revised 20120101

Intelligent Control Laboratory, Peking University

Jan. 2012

Index

1. Summary

Please download the latest version of the guide from the official website of China Underwater Robot Games (http://robot.pku.edu.cn).

1.1. Target Reader

The guide is provided for the potential designer, developer, tester and maintainer of the software URWPGSim2D and the strategy programmer who participates the 2D simulation game of China Underwater Robot Games. All the technical details related to URWPGSim2D are kept here as many as possible.

1.2. Glossary

1. URWPGSim2D：2 Dimension Edition of Underwater Robot Water Polo Game Simulator
2. MRDS：Microsoft Robotics Developer Studio
3. CCR：Concurrency and Coordination Runtime, a technical and base software library to solve the concurrency problem in the development of robot software.
4. DSS：Decentralized Software Services，a technical and base software library to solve the asynchronous problem in the development of robot software.
5. Simulation Mission：Mission, a simulation game or experimental project.
6. Simulation Environment：SimEnvironment，the virtual environment for the simulation mission, including simulation field(the pool for a simulation game or experiment), simulate water polo(none or several), simulate rectangular obstacle(none or several), simulate round obstacle(none or several).
7. Simulation RoboFish：RoboFish，the robotic fish for a simulation game or experiment.
8. Simulation Loop：the course of executing all the simulation actions sequently for one time
9. Simulation Period：the time for the simulation loop
10. Simulation Action：the simulation actions include the decisions for the robotic fish, the kinematical computation of all the moveable objects, the collision process of all the objects.

2. Development Environment

2.1. Hardware Environment

URWPGSim2D can be developed on the PC or workstation with the hardware configuration requirements shown in table 2-1.

Table 2-1 Hardware Configuration for running URWPGSim2D

Main Components	Minimum Configuration	Recommended Configuration

CPU	Intel P4 2.0GHz or similar AMD CPU	Intel E7300 2.66GHz or higher
Memory	256MB	2GB or more
Video Card	Support DirectX 9.0，Pixel Shader 3.0，video memory128M or more	
Harddisk	10GB	80GB or more

2.2. Software Environment

OS：Windows XP Professional SP3，Windows Vista or Windows 7。

.Net Framework：.Net Framework 3.5 with SP1。

IDE：Microsoft Visual Studio Team System 2008 Team Suite with SP1，or Microsoft Visual Studio 2008 Professional with SP1。

Program Language：C# V3.0。

MRDS：Microsoft Robotics Developer Studio 2008 R3。

Accessory：Microsoft XNA Framework Redistributable 3.1，Microsoft Excel 2003 Com Library。

2.3. Set Up Standard Development Environment

The software for setting up standard development environment are provided on the official website of Underwater Robot Game Of China.

1. Install Windows XP Professional SP3 on the PC or workstation.
2. Install DotNet3.5SP1 and XNA3.1 with the default setting.
3. Install Microsoft Visual Studio Team System 2008 Team Suite and SP1 with the relevant components for C# development.
4. Install TortoiseSVN1.6.5和VisualSVN package of VS2008 with the default setting.

2.4. Start Development

Get the SVN address of URWPGSim2D source code and corresponding username and password from the official website of Underwater Robot Game Of China. Select “Get Solution from Subversion” under the menu “VisualSVN” in VS2008, input the SVN address and input the username and password in the pop-up dialog box to get the latest version of source code.

To facilitate the debugging with the shortcut key in Visual Studio, ConductorSvr need be set as startup project.

2.5. Supplement

MRDS need not to be installed. All the DLL files of MRDS used in URWPGSim2D have been included in the directory URWPGSim2D\bin of the source code package.

3. General Design

3.1. Functional Design

URWPGSim2D is mainly used as the platform of underwater robot games and scientific research to facilitate the extension of the game and experimental project and the independent development of strategy algorithm.

URWPGSim2D includes two parts, URWPGSim2DServer and URWPGSim2DClient. URWPGSim2DServer simulates the underwater environment, controls and shows the simulation and result, sends real-time simulation environment and process information. Semi-distributed client simulates the team of underwater robots, while fully distributed client simulates one underwater robot, both loading the strategy of the game or experiment, complete the process of decision computation and sending decisions to the server.

3.2. VS2008 Solution & Project Struction

URWPGSim2D is generated from the VS2008 C# Solution name “URWPGSim2D.sln”, which includes at least 9 projects (possible to be extended) and the generating order decided by their dependence, is shown as Table 3-1. All the projects have the key file named “URWPGSim2D.snk” for the signature and generate a program set with strong names.

URWPGSim2D1.0.11.31 includes 9 private components, MRDS and other third-party components.

Table 3-1 URWPGSim2D Private Components

Component	Corresponding Files	Description	Corresponding Project	Dependent Components
Core	URWPGSim2D.Core.dll	Kinematics and dynamics computation module; random disturbance module; collision response module	Core	none
Common	URWPGSim2D.Common.dll 、config.xml	Simulation robotic fish module; simulation environment (field, water polo, obstacle, channel, etc); simulation mission module; collision detection module; system configuration module; auxiliary function module	Common	Core
Match	URWPGSim2D.Match.dll	Specific simulation mission realization module (adding	Match	Core 、Common

		game projects)		
StrategyLoader	URWPGSim2D.StrategyLoader.dll	Auxiliary module for realization of loading strategy dynamically	StrategyLoader	Common
Gadget	URWPGSim2D.Gadget.dll	Auxiliary module for drawing track and displaying real-time information	Gadget	none
Sim2DSvr	Sim2DSvr.manifest.xml 、URWPGSim2D.Sim2DSvr.Y2010.M11.dll 、URWPGSim2D.Sim2DSvr.Y2010.M11.Proxy.dll 、URWPGSim2D.Sim2DSvr.Y2010.M11.Transform.dll	DSS services and interface module for the server	Sim2DSvr	Common 、Match 、StrategyLoader、Gadget
Sim2DClt	Sim2DClt.manifest.xml 、URWPGSim2D.Sim2DClt.Y2010.M11.dll 、URWPGSim2D.Sim2DClt.Y2010.M11.Proxy.dll 、URWPGSim2D.Sim2DClt.Y2010.M11.Transform.dll	DSS services and interface module for the client	Sim2DClt	Common 、Sim2DSvr 、StrategyLoader
ConductorSvr	URWPGSim2DServer.exe	Boot loader for the server	ConductorSvr	Sim2DSvr
ConductorClt	URWPGSim2DClient.exe	Boot loader for the client	ConductorClt	Sim2DClt

The folder of URWPGSim2D Solution is named URWPGSim2D in default and can be

renamed as you wish. The Solution includes 9 projects, all the folders of projects are put under the folder of Solution. Besides, there are two more folders, URWPGSim2D for output and Strategy for strategies.

In the deployment design, URWPGSim2D is totally green without changing the registry and copying any file to the system folder. All the third-party components needed are put in the program folder and can be used immediately after being copied. The output folder of URWPGSim2D has a subfolder “bin” for running the program. The folder “bin” can not be renamed because of the operation mechanism of MRDS DSS service. The DSS service component generates .dll files which are deployed in the subfolder “bin” of the installation folder of MRDS and loaded by DssHost.exe in the folder “bin”. When loading, DssHost.exe will search for the folder “store” among the superior folders of “bin” to query for the relevant Cache information. If none, a new folder will be created automatically. Thus there must be a subfolder “bin” in the output folder “URWPGSim2D”.

The output folder of all the projects is set as “..\URWPGSim2D\bin”, in which all the third-party components are put. The property of “Copy Local” in the referenced assembly (one .dll file here) of all the projects is set as “False”. So the channeles of the referenced assembly of all the projects are fixed to the relative directory “..\URWPGSim2D\bin”. The reference will be updated in time for any new versions.

The folder “Strategy” is for Strategy Solution, which provides most strategy module for simulation missions. To program the strategy of simulation game or self-designed experiment, you can use the strategy of old project directly or add the strategy of new project.

3.3. Overall Structure

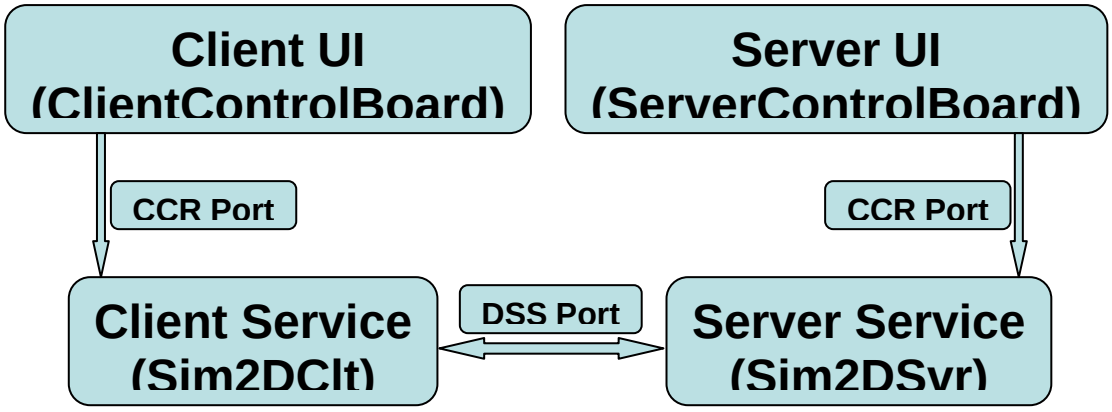

Figure 3-1 URWPGSim2D Overall Structure

3.4. Server Structure

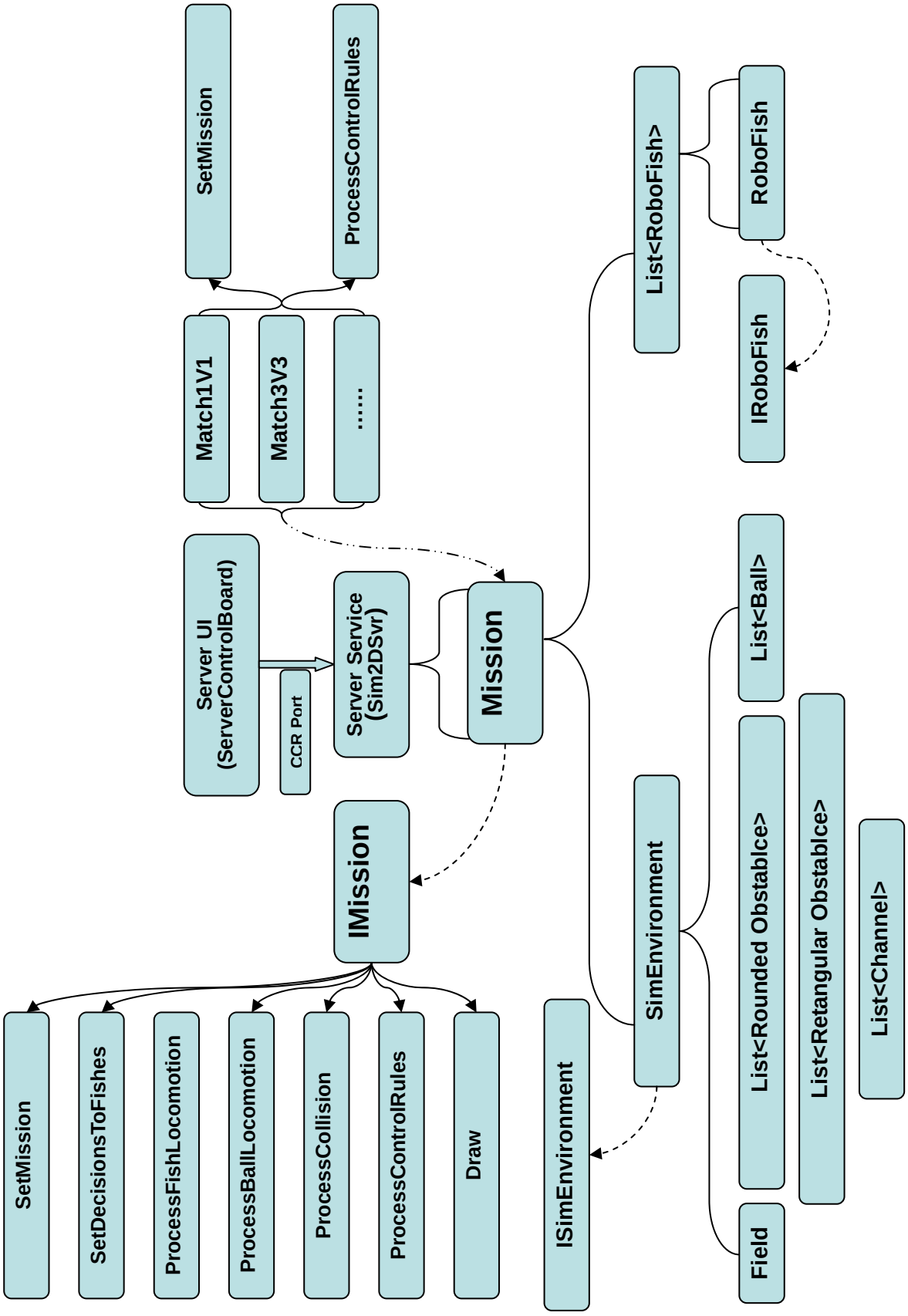

Figure 3-2 URWPGSim2D Server Module Structure

3.4.1. Simulation Mission As the Center

URWPGSim2D module structure is centralized with simulation mission as shown in figure 3-2.

URWPGSim2D is designed with object-oriented thought. From the respect of object modeling, there are 3 models including simulation robotic fish, simulation environment and simulation mission (game or experiment project) and simulation mission is the center. The simulation mission includes the team list of simulation robotic fish and simulation environment. All 3 models have top models respectively, base class RoboFish of simulation robotic fish, SimEnvironmen of simulation environment and Mission of simulation mission. Class RoboFish defines common characteristics (represented by class properties) and common behaviors (represented by class method) of simulation robotic fish needed in specific simulation mission. Class SimEnvironmen defines common characteristics of simulation environment needed in specific simulation mission. Class Mission defines common characteristics and common behaviors of specific simulation mission.

The common behaviors of simulation robotic fish and simulation environment are defined rarely or unnecessarily, while those of simulation mission are defined a lot. Though the base class RoboFish and SimEnvironment realize the interfaces IRoboFish and ISimEnvironment in form, there is no defined method on both interfaces. The base class Mission realizes the interface IMission with a number of defined methods. Some methods can be realized directly in the base class Mission, while the other only can be provided virtual functions and realized by override in the specific simulation mission class.

A generic class Team<TFish> is defined for the formation of robotic fish. In TFish, there are one list （List<TFish>）member Fishes for saving all robotic fish objects of the team and one team common parameter（TeamCommonPara） member Para for saving all characteristic parameters of the team (e.g. team name, number of simulation robotic fish, present score).

The simulation environment includes several elements, such as simulation field, simulation water polo, simulation obstacle and simulation channel. In the base class of simulation environment SimEnvironment, there are one simulation field (Field) class member FieldInfo for saving simulation field objects, one simulation water polo (Ball) class list（List<Ball>） member Balls for saving all simulation water polo objects in current simulation environment, one simulation round obstacle (RoundedObstacle) class list （List<RoundedObstacle>） member ObstaclesRound for saving all simulation round obstacle in current simulation environment, one simulation rectangular obstacle (RectangularObstacle) class list （List<RectangularObstacle>） member ObstaclesRect for saving all simulation rectangular obstacle in current simulation environment, and one simulation channel (Channel) class list （List<Channel>）member Channels for saving all simulation channel objects in current simulation environment.(Note: In fact, the concept of simulation channel has never been used and can be substitute by two paralleled simulation rectangular obstacles. So the concept of simulation channel is abandoned.)

Simulation mission class Mission includes one simulation robotic fish base class RoboFish team list （List<Team<RoboFish>>） member TeamsRef, one simulation environment base class

SimEnvironment member EnvRef and one simulation mission common parameter MissionCommonPara class member CommonPara.

Specific simulation mission class (e.g. simulation mission class of 3v3 match Match3V3) inherits base class Mission. Correspondingly, specific simulation robotic fish class (e.g. Fish3V3) inherits base class RoboFish and specific simulation environment class (e.g. Fish3V3) inherits base class SimEnvironment.

3.4.2. Simulation Period As the Main Line

After simulation mission start running, simulation period will continue periodically unless set simulation time runs out or paused or stopped manually or by the program.

The simulation period will take all simulation actions according to the following procedure.

1. To allocate decision valueSetDecisionsToFishes;
2. To compute kinematic parameters ProcessFishLocomotion of simulation robotic fish;
3. To compute kinematic parameters ProcessBallLocomotion of simulation water polo;
4. To process collision detection and response ProcessCollision of all objects in the field;
5. To process specific rules ProcessControlRules of current simulation mission, such as foul and score;
6. To process local and remote strategy calling;
7. To process interface dynamic data update.

3.5. Version Control

3.5.1. Source Code Management

In the development of URWPGSim2D, SVN is used for source code version control. SVN server installs VisualSVN-Server-2.0.7, while SVN client installs TortoiseSVN-1.6.5（TortoiseSVN-1.6.5.16974-win32-svn-1.6.5） with Visual Studio 2008 package VisualSVN-1.7.6.

3.5.2. Product Management

URWPGSim2D product version number includes 4 sections “Majo.Minor.[Revision[.Build]]”. Major represents primary version number and starts from 1, which will be added by 1 for each time when there are great changes of products function. Minor represents secondary version number and starts from 0, which will be added by 1 for each time when there are big changes of products function. Revision represents revision number, which is the Revision number of source code library of SVN server. Build is the generation date of 6 digits, made of YYMMDD. For example, 1.1.103.110713 represents the product version generates from the source code of Revision number 103 in July 13, 2011 with no great change and one big change of function.

4. Strategy Program

4.1. What is Strategy

In terms of content, strategy is the algorithm code used to control simulation robotic fish participating in current simulation mission. In terms of form, strategy is a .dll file able to load in the server and client.

4.2. How to Call the Strategy

4.2.1. Local Mode

Select “Local” from “Referee→Strategy” in the interface of the server URWPGSim2DServer. Under local mode, the strategies of all teams in simulation mission are loaded in the server and run with URWPGSim2DServer.exe in the same process space. As simulation period runs in the process space of URWPGSim2DServer.exe, all strategies and simulation period run in the same process space. Therefore all control instructions of simulation robotic fish and operation instructions of simulation period of all teams are from the same process. So, local mode is a centralized simulation mode.

4.2.2. Remote Mode

Select “Remote” from “Referee→Strategy” in the interface of the server URWPGSim2DServer. Under remote mode, the number of URWPGSim2DClient.exe processes must equal to the number of teams in the simulation mission for loading the strategy of each team. These processes can run in the same computer or different computers within one LAN (theoretically workable in WAN, but not tested yet). The strategy of each team and URWPGSim2DClient.exe process that loads it are running in the same process space. Therefore the control instructions of all simulation robotic fish of each team are independent to simulation period operation instructions. The control instructions of simulation robotic fish of all teams are independent to each other, while those of one team are from the same process. So, remote mode is a semi-distribution simulation mode.

If process URWPGSim2DClient.exe and URWPGSim2DServer.exe are running in different computers, the configuration “<dssp:Service>http://**localhost**:50000/Sim2DSvr</dssp:Service>” in the file “../URWPGSim2D/bin/**Sim2DClt.manifest.xml**” of the computer that URWPGSim2DClient.exe is running needs to be changed from localhost to the IP or hostname of the computer that URWPGSim2DServer.exe is running on.

4.2.3. Asynchronous Call

The asynchronous manner is used in both local mode and remote mode.

In local mode, the calling code is in the method Sim2DSvrService.NextStepProcessDetail that the process URWPGSim2DServer.exe of the server launches one Arbiter.Receiver for each team to call the thread in CCR threadpool asynchronously and process decision algorithm for decision value. GetLocalDecision and simulation period are asynchronous. The process result is that decision array is filled in the public space DecisionRef and simulation period distributes

recent decision value from DecisionRef to corresponding robotic fish. With regard to the simulation mission of two or more teams involved, the process order of GetLocalDecision for each team in one simulation period is uncertain.

In remote mode, complicated CCR and DSS communication process are related. Call entry code is in the method Sim2DSvrService.NextStepProcessDetail that the process URWPGSim2DServer.ex of the server calls Sim2DSvrService.MissionParaNotification asynchronously by SpawnIterator and informs Mission object values of current simulation mission to all clients which includes all simulation environment and process information. After service instance Sim2DCltServic of process URWPGSim2DClient.exe on the client receives MissionParaNotification, it calls Sim2DCltService.AnnounceDecisionToServer asynchronously by SpawnIterator and process to get decision array which will be sent to the server as ClientAnnounceDecision message. After service instance Sim2DSvrService of process URWPGSim2DServer.exe on the server receives ClientAnnounceDecision, CRC scheduler will call ClientAnnounceDecisionHandler with threads in the thread pool asynchronously and fill the received decision array in the public space DecisionRef for the simulation period to distribute to corresponding simulation robotic fish by SetDecisionsToFishes. The aforementioned course does not include that the client initiate to connect the server and request for establishing Subscribe/Notify relation with the server after booting. There is another introduction on the communication details related to URWPGSim2D.

4.3. Program Guide

以下描述中%URWPGSim2D%为 URWPGSim2D Solution Directory，如 D:\My Documents\Visual Studio 2008\Projects\URWPGSim2D。

In the following description, %URWPGSim2D% means URWPGSim2D Solution Directory, e.g. D:\My Documents\Visual Studio 2008\Projects\URWPGSim2D.

4.3.1. About Program

To program a strategy, you need creat a **VS2008** project of Windows Class Library type and corresponding Solution with C# program language based on **.Net Framework 3.5** first.

The namespace of the project must be **URWPGSim2D.Strategy**. There must be a class named **Strategy**, which must inherit **MarshalByRefObject class** and support **IStrategy interface** for realizing **GetTeamName** and **GetDecision** method. The specific information of the interface refers to the program structure in the 7th items. At least 4 Reference, Microsoft.Dss.Base.dll, Microsoft.Xna.Framework.dll, URWPGSim2D.Common.dll and URWPGSim2D.StrategyLoader should be added to the project.

1. Strategy Solution Strategy.sln in the folder of %URWPGSim2D%\Strategy\ can be used directly. Either strategy project is added to Strategy Solution, or existing Strategy template Project can be used directly.

2. New Solution can be created with new strategy Project (e.g. Strategy3VS3). The Project should be put in the subfolder of Solution which can be put anywhere with any name but recommended name is Strategy.

3. Modify Project properties. After Strategy Project is created, e.g. Strategy3VS3 for Match3VS3 simulation mission, some Project properties need to be modified. In Solution Browser, select Strategy Project, e.g. Strategy3VS3, click right mouse button and select Properties. In popup page, change the option Application→ Default namespace to URWPGSim2D.Strategy.

4. Modify Class name. Rename default cs file , e.g. Class1.cs, to StrategyProjectName.cs, e.g. Strategy3VS3.cs. Reopen the file, and rename namespace, e.g. ClassLibrary1, to URWPGSim2D.Strategy and class, e.g. Class1 to Strategy:MarshalByRefObject, IStrategy.

5. Class Strategy must reload MarshalByRefObject 的 InitializeLifetimeService method and realize with one statement “return null” to avoid timeout problems when Strategy object is loaded outside the field of main program. Otherwise, Strategy object will be destroyed automatically if there is a badly long time-out. After the time-out, the strategy does not exist.

6. The instance of library Strategy will be stored in the memory after loaded only if strategy is changed during the simulation mission. Thus, private variants can be defined to save necessary information.

7. Recommended program structure is as the following.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

using xna = Microsoft.Xna.Framework;
using URWPGSim2D.Common;
using URWPGSim2D.StrategyLoader;

namespace URWPGSim2D.Strategy
{
    public class Strategy : MarshalByRefObject, IStrategy
    {
        #region reserved code never be changed or removed
        /// <summary>
        /// override the InitializeLifetimeService to return null instead of a valid ILease implementation
        /// to ensure this type of remote object never dies
        /// </summary>
        /// <returns>null</returns>
        public override object InitializeLifetimeService()
        {
            return null; // makes the object live indefinitely
        }
        #endregion
```

```
        /// <summary>
        ///  决策类当前对象对应的仿真使命参与队伍的决策数组引用  第一次调用GetDecision时分配空间
        /// </summary>
        private Decision[] decisions = null;

        /// <summary>
        /// get team name: set team name here
        /// </summary>
        /// <returns>teamname</returns>
        public string GetTeamName()
        {
            return "3VS3 Test Team";
        }

        /// <summary>
        /// get the decision data array of current simulation mission (game) and  获取当前仿真使命（比赛项目）当前队伍所有仿真机器鱼的决策数据构成的数组
        /// </summary>
        /// <param name="mission">服务端当前运行着的仿真使命 Mission 对象</param>
        /// <param name="teamId">当前队伍在服务端运行着的仿真使命中所处的编号
        ///  用于作为索引访问 Mission 对象的 TeamsRef 队伍列表中代表当前队伍的元素</param>
        /// <returns>当前队伍所有仿真机器鱼的决策数据构成的 Decision 数组对象</returns>
        public Decision[] GetDecision(Mission mission, int teamId)
        {
            //  决策类当前对象第一次调用 GetDecision 时 Decision 数组引用为 null
            if (decisions == null)
            {//  根据决策类当前对象对应的仿真使命参与队伍仿真机器鱼的数量分配决策数组空间
                decisions = new Decision[mission.CommonPara.FishCntPerTeam];
            }

            #region decision computation that is realized by your team
            #endregion
            // codes start from here.

            #endregion

            return decisions;
        }
    }
}
```

8. GetTeamName interface is used to set team name by substituting “3VS3 Test Team” with your team name.

9. GetDecision interface is used to generate decision data of Strategy objects for all simulation

robotic i-th fishn corresponding team. If the decision-making is complicate, it is suggested to encapsulating various subprocedure to Private method of Class Strategy which can be called by GetDecision or subprocedure.

10. Since class Strategy called 2 components, URWPGSim2D.Common.dll and URWPGSim2D.StrategyLoader, both of which have strong name (see MSDN for details about strong name). So if main program URWPGSim2D (including 2 aforementioned components) updates, dll file for Strategy must be added to corresponding class Strategy again and regenerated before normal loading.

4.3.2. About Business

1. The direct target of strategy programming is to fill in current team decision array “decisions” with decision values, i.e., to generate control instructions for all simulation robotic fish of the team in simulation mission corresponding to current objects of class Strategy. The control instruction of simulation robotic i-th fishs a value of Decision type.

2. Data type Decision includes 2 int members: VCode for velocity gear and TCode for turning gera.

3. VCode values range from 0 to 14, 15 integers in total. Each integer corresponds to one velocity, which increases but not strictly. According to speed data from experiments, the velocity of some gear is slower than lower one.

4. TCode values range from 0 to 14, 15 integers in total. Each integer corresponds to one angular velocity. The integer 7 corresponds to swimming straight without any angular velocity, while 6 to 0 and 8 to 14 corresponds to turning left and right respectively with increasing angular velocity.

5. Any gear shift of velocity or turning needs several simulation periods to realize stable linear or angular velocity. At present, kinematics computation decides that stable linear or angular velocity is close to but less than the value of corresponding gear.

6. Simulation robotic i-th fishs prohibited to spin on the spot, i.e. it is impossible that linear velocity of simulation robotic i-th fishs zero while angular velocity is not. The platform has set a rule that if linear velocity is slower than stable velocity of gear 1, angular velocity will be se to zero forcefully. Don’t reduce linear velocity of simulation robotic fish to 0 and only give it angular velocity for turning around, which will make it stay still.

7. The field coordinate system, point and vector are defined as follows. The origin of the coordinate plane is the center of the rectangular field with right as plus X axis and down as plus Z axis. From plus X axis to minus X axis, the angle varies from **0 to π clockwise and from** 0 to -π counterclockwise. In consideration of the consistence with 3-dimensional coordinate system of MRDS, the horizontal plane is represented by XOZ, while Y axis is the third dimension. The definitions of vector and point are of Vector3 type in library XNA with X and Z dimensions in use and Y dimension set to 0. In the transition of 2-dimensional points and vectors to 3-dimensional ones, X in 2D corresponds to X in 3D while Y in 2D to Z in 3D.

8. The strategy of antagonistic simulation mission must decide its own half and target goal. The behavior and related decision algorithm of simulation robotic fish are designed according to target goal.

9. The integer-type parameter teamID of GetDecision interface represents the serial number of the team corresponding to current Strategy object in simulation mission running on the server (starting from 0). The initial value of teamID is decided by the order of loading strategy in Local mode and the sequence of client launching in Remote mode. After 2 teams in antagonistic simulation mission change ends, teamID will be exchanged by half handling code of the server or client strategy calling entry and transmitted to GetDecision.

10. The mission-type parameter mission of GetDecision interface include all public information of current simulation mission running on the server for strategy. mission. TeamsRef[teamId], pointing to Team<RoboFish> objects of the team in corresponding simulation mission of current strategy objects, can be used for visiting all public information of the team and its simulation robotic fish. In an antagonistic simulation mission of 2 teams, mission.TeamsRef[(1 + teamId) % 2] which points to Team<RoboFish> objects of opponent team, can be used to visit all public information of the other team and its simulation robotic fish.

11. Table 4-1 includes all parameters which can be used in the strategy. The reference of all members starts with "mission.", e.g. mission.CommonPara.TeamCount, among which members of "CommonPara.*" class represent the names of several following members, e.g. CommonPara.MsPerPeriod. The complete reference is mission.CommonPara.MsPerPeriod. There are other types of members, such as MissionCommonPara, Team<RoboFish>, TeamCommonPara, RoboFish, Field, RetangularObstacle, RoundedObstacle, which are not open to strategy but related to server processing. Do not use them.

Table 4-1 All parameters for Strategy

Member name	Type	Meaning
CommonPara	MissionCommonPara	Public parameter of current simulation mission
CommonPara.*		
FishCntPerTeam	int	Number of simulation robotic fish of each team
MsPerPeriod	int	Milliseconds of simulation period
RemainingPeriods	int	Simulation periods left
TeamCount	int	Number of teams in current simulation mission
TotalSeconds	int	Seconds of running time in current simulation mission
TeamsRef	List<Team<RoboFish>>	Team list of current simulation mission
TeamsRef[teamId]	Team<RoboFish>	Team in the simulation mission corresponding to current object of decision class (current team)
TeamsRef[teamId].*		
Para	TeamCommonPara	Public parameter of current team

Member name	Type	Meaning
Fishes	List<RoboFish>	Simulation robotic fish list of current team
TeamsRef[teamId].Para.*		
FishCount	int	Number of simulation robotic fish of current team （equaling to CommonPara.FishCntPerTeam）
MyHalfCourt	HalfCourt（enum）	Half field of current team （HalfCourt.LEFT(0) / Halft.RIGHT(1)）
Score	int	Current team score
Name	string	Current team name
TeamsRef[teamId].Fishes[i].*		
PositionMm	xna.Vector3	Position of i-th fish of current team (center of fish fore-end rigid body)
BodyDirectionRad	float	Orientation of i-th fish of current team(direction of long side of fish fore-end rigid body)
VelocityMmPs	float	Velocity of i-th fish of current team
VelocityDirectionRad	float	Velocity direction of i-th fish of current team (the same as BodyDirectionRa)
AngularVelocityRadPs	float	Angular velocity of i-th fish of current team
BodyLength	int	Fore-end rigid body length of i-th fish of current team
BodyWidth	int	Fore-end rigid body width of i-th fish of current team
CollisionModelRadiusMm	int	Radius of external round model of i-th fish of current team in collision detection
CollisionModelBodyRadiusMm	int	Radius of circumcircle of fore-end rectangular rigid body of i-th fish of current team
CollisionModelTailRadiusMm	int	Radius of tail round model of i-th fish of current team in collision detection
PolygonVertices[0]	xna.Vector3	Position of head of i-th fish of current team
EnvRef	SimEnvironment	Environment object of current simulation mission
EnvRef.*		
FeildInfo	Field	Simulation field object of current simulation mission
Balls	List<Ball>	All simulation water polo objects of current simulation mission
Balls[i]	Ball	i-th simulation water polo of current simulation mission
ObstaclesRect	List<RetangularObstacle>	List of all simulation rectangular obstacles of current simulation mission
ObstaclesRect[i]	RetangularObstacle	i-th simulation rectangular obstacle object of current

Member name	Type	Meaning
		simulation mission
ObstaclesRound	List<RoundedObstacle>	List of all simulation round obstacles of current simulation mission
ObstaclesRound[i]	RoundedObstacle	i-th simulation round obstacle object of current simulation mission
EnvRef.FieldInfo.*		
FieldLengthXMm	int	Length of current simulation field in X direction
FieldLengthZMm	int	Length of current simulation field in Z direction
GoalDepthMm	int	Goal depth of current simulation field（length in X direction）
GoalWidthMm	int	Goal width of current simulation field（length in Z direction）
ForbiddenZoneLengthXMm	int	Length in X direction of penalty area of current simulation field
ForbiddenZoneLengthZMm	int	Length in Z direction of penalty area of current simulation field
LeftMm	int	Coordinate X of left bound of current simulation field
RightMm	int	Coordinate X of right bound of current simulation field
TopMm	int	Coordinate Z of upper bound of current simulation field
BottomMm	int	Coordinate Z of bottom bound of current simulation field
EnvRef.Balls[i].*		
PositionMm	xna.Vector3	Position of i-th water polo in current simulation mission
RadiusMm	int	Radius of i-th water polo in current simulation mission
VelocityMmPs	float	Velocity of i-th water polo in current simulation mission
VelocityDirectionRad	float	Direction of i-th water polo in current simulation mission
EnvRef.ObstaclesRect[i].*		
PositionMm	xna.Vector3	Position of i-th rectangular obstacle of current simulation mission
LengthMm	int	Length of i-th rectangular obstacle of current simulation mission
WidthMm	int	Width of i-th rectangular obstacle of current simulation mission
DirectionRad	float	Direction of i-th rectangular obstacle of current simulation mission (length orientation)
EnvRef.ObstaclesRound[i].*		
PositionMm	xna.Vector3	Position of i-th round obstacle of current simulation mission

Member name	Type	Meaning
RadiusMm	int	Radius of i-th rectangular obstacle of current simulation mission
HtMissionVariables	Hashtable	Hash table of special parameters for strategy in current simulation mission, with key string as variant name and value string as variant value

12. The special parameters of particular simulation mission (e.g. flag CompetitionPeriod indicating current game phase in antagonistic game like water polo 3vs3) can be get by HtMissionVariables["key name"] and converted to original data type value by Convert.To*** for use. For example, current game phase value of water polo 3vs3 can be get by: int matchPeriod = Convert.ToInt32(mission.HtMissionVariables["CompetitionPeriod"]). The special parameters of each particular simulation mission that are transmitted to strategy are determined by simulation mission designers and their meaning and use are described in simulation mission rule document.

4.3.3. About Debugging

1. The strategy can be debugged under Local mode first for basic logistics and then take a fitness test and parameter amendment under Remote mode. The game strategy must be tested under Remote mode. The strategy for antagonistic game must test whether it works as expected after two team change ends during halftime.

2. For convenience sake, it is needed to set Debug→Start action→Start external program option as %URWPGSim2D%\URWPGSim2D\bin\URWPGSim2DServer.exe for debugging under Local mode and %URWPGSim2D%\URWPGSim2D\bin\URWPGSim2DClient.exe for debugging under Remote mode in Project Properties. Note: the code downloaded from SVN server does not include these two files under bin folder, so do other original components except URWPGSim2D.Core.dll before clicking Build Solution first.

3. Since one Solution can only set one launching game with several strategy Projects, it is necessary to right-click strategy Project name for debugging and select D**ebug →Start new instance.**

4. To debug the strategy for simulation mission including 2 or more teams, it is necessary to generate one or more backup strategy. The debugged strategy can only be loaded for one team per time. Otherwise, the breakpoint in strategy code will recur several times in one period and go against tracking.

5. When the error like failing to find components occurs in debugging, the quoting of debugged projects should be checked first. If it is incorrect, you should delete it and add another from %URWPGSim2D%\URWPGSim2D\bin\ (right-click “quoting”, select “add quoting” →”browse”, find the aforementioned folder and choose correct program assembly). If the version of main program has been updated and dll files of strategy have been regenerated, it

may require deleting the original quoting and add again, which depends on whether the original folder of quoting program assembly is the same as new one.

5. Standard Function

5.1. PoseToPose Function

The function belongs to Helpers type (static type) of URWPGSim2D.StrategyHelper namespace.

5.1.1. Introduction of Function

PoseToPose function is the control function from one pose to another which can realize the precise control of simulation robotic fish from current pose to target pose. The control of PoseToPose can be devided to 2 stages. In the first stage, the fish is controlled to swim to temporary target point rapidly. In the second stage, the fish is controlled to swim to target point. Therein, temporary target point is on the reverse extension line and the distance threshold can be controlled.

5.1.2. Description of Function Parameters

5.1.2.1. Function Prototype

```
public static void PoseToPose(ref Decision decision, RoboFish fish, xna.Vector3 destPtMm, float destDirRad, float angThreshold, float disThreshold, int msPerPeriod, ref int times)
```

5.1.2.2. Parameter Description Table

Table 5-1 PoseToPose function parameter description table

Parameter Name	Parameter Type	Parameter Description
Decision	ref Decision	Decision variant value (output parameter) computed by PoseToPose
Fish	RoboFish	Simulation robotic fish object which runs PoseToPose
destPtMm	xna.Vector3	Coordinates of target position (target point)
destDirRad	Float	Radian value of target orientation（target orientation）
angThreshold	Float	Upper limit of key controlling parameter (absolute value of the difference between middle direction and orientation of fish body), 30 degrees in default
disThreshold	Float	Threshold of key controlling parameter (distance between temporary target point and final target point)
msPerPeriod	Int	Milliseconds of each simulation period, i.e. mission.CommonPara.MsPerPeriod
Times	ref int	See 5.1.3（output parameter）

5.1.3. Calling Method

1. Add integer member variant of Strategy type in codes with the initial value 0 as the input of last parameter times when calling PoseToPose function, e.g. int times, which is used to

record the time in stage 2 of algorithm.

2. In order to call PoseToPose in member function (method) of Strategy type in codes, the following calling codes with recommended parameters can be used and adjusted according to the actual debugging situation.

```
StrategyHelper.Helpers.PoseToPose(ref decisions[i], mission.TeamsRef[teamId].Fishes[i], targetPoint, targetDirection, 30.0f, 8 * b.RadiusMm, mission.CommonPara.MsPerPeriod, ref times);
```

3. The caller needs to decide whether the pose control target is reached by oneself. After reaching pose control target once, you need to initial the input invariant (member variant of Strategy type) of parameter times defined in the first step for one more calling.

5.2. Dribble Function

The function belongs to Helpers type (static type) of URWPGSim2D.StrategyHelper namespace.

5.2.1. Introduction of Function

Dribble function is used to realize the dribbling control of simulation robotic fish in some games.

5.2.2. Description of Function Parameters

5.2.2.1. Function Prototype

```
public static void Dribble(ref Decision decision, RoboFish fish, xna.Vector3 destPtMm, float destDirRad,float angleTheta1, float angleTheta2, float disThreshold, int VCode1, int VCode2, int periods, int msPerPeriod, bool flag)
```

5.2.2.2. Parameter Description Table

Table 5-2 Dribble function parameter description table

Parameter Name	Parameter Type	Parameter Description
Decision	ref Decision	Decision variant value (output parameter) computed by Dribble
Fish	RoboFish	Simulation robotic fish object which runs Dribble
destPtMm	xna.Vector3	Coordinates of target position (target point)
destDirRad	float	Radian value of target orientation（target orientation）
angleTheta1	float	Fisrt threshold of the angle between fish body and target orientation. Below the threshold, a reasonable speed gear will be given to simulation robotic fish. (see the description of parameter disThreshold)
angleTheta2	float	Second threshold of the angle between fish body and target orientation. Below the threshold, simulation robotic fish will swim straight. Above the threshold, simulation robotic fish will change swimming direction.
disThreshold	float	Threshold of distance. Above the threshold, simulation

Parameter Name	Parameter Type	Parameter Description
		robotic fish will swim with speed gear VCode1. Below the threshold, simulation robotic fish will swim with speed gear VCode2.
VCode1	int	Swimming gear 1 (6 gears in default)
VCode2	int	Swimming gear 2 (4 gears in default)
Periods	int	Number of periods needed for switching between speed and turning gear with recommended value range from 5 to 20 which can avoid robotic fish from turning too much.
msPerPeriod	int	Milliseconds of each simulation period, i.e. mission.CommonPara.MsPerPeriod
Flag	bool	Coordinate standard of robotic fish, with the value true as PositionMm, i.e. the center of fish body and false as PolygonVertices[0], i.e. fish head.

5.2.3. Calling Method

In order to call Dribble in member function (method) of Strategy type in codes, the following calling codes with recommended parameters can be used and adjusted according to the actual debugging situation.

```
StrategyHelper.Helpers.Dribble(ref decisions[i], mission.TeamsRef[teamId].Fishes[i], targetPoint, targetDirection, 5, 10, 150, 6, 4, 15, 100, true);
```