SoccerCode: A Game System for Introductory Programming Courses in Computer Science

Minghao Wang , Xiaolin Hu

Abstract—**How to attract students' interests in order to increase the rate of their enrollment in studying computer science/engineering is an important topic. An introductory programming course is the most direct and sensitive interface where students are able to explore the interests and research potential in this field. Therefore, it is important to develop appropriate teaching strategies to avoid making this course tedious. Serious games are widely recognized as an attractive approach for training and education recently. In this paper, we present SoccerCode, a serious game-based system for teaching introductory programming courses in computer science. The inspiration of the project, the basic architecture and functionalities are introduced in the paper. Future work includes technical improvement such as migrating to web-based system and implementing it systematically in CS introductory courses.**

Index Terms—**educational game, serious game, robot soccer, introductory programming, agent-based modeling.**

I. Introduction

IN Computer Science education, an important question is how to attract new students and motivate their effort in further study on this discipline. According to a report published on May 2011(and conducted in 2010) [7], although the overall declarations and productions of CS/CE bachelor degrees in U.S universities increased from that in 2009, they still keep in a level of having greatly dropped down compared to the data from 1997 to 2004. One critical cause of this reduction is that as a discipline, computer science/engineering nowadays has been focusing more on solving practical matters rather than exploring the traditional excitement of attempting to mimic/magnify intelligence through computers. As a result, computer science has become less and less attractive to newly enrolled college students, comparing to disciplines such as biology or chemistry which keeps challenging great questions in nature, science and human---ourselves. Therefore, to encourage students to enroll in this field, more efficient educating strategies are necessary. Traditionally, universities tend to give an introductory programming course as one of the first several courses for students who are interested in CS/CE major. This kinds of courses aim at giving students an overview of computer science including programming language, algorithm and data structure [4], and is the most direct and sensitive interface for new students to understand the research challenges in CS/CE. Therefore, it is important to avoid making the introductory course tedious; the strategy must be attractive enough to let new students realize the numerous possibilities of CS/CE discipline but at the same time maintains the integrity and rigorousness of all the necessary traditional components.

The popularity of video games provides appropriate pathways to develop new teaching approach. In fact, games in education have recently been widely accepted to be effective for motivating students' interests. In their research, Leutenegger and Edgington has investigated the approach to introduce game developing in introductory programming course for attracting and lagging students interests.[6]. Similar approach has been applied to Lorenzen and Becker's researches [9]. In another research, Ranum [2] and his cooperators introduced several successful approaches to teach multi-media manipulation in python as introductory programming course. All the attempts listed above have improved the attraction of the courses comparing to traditional teaching strategies.

While most of existing teaching programs focus on the programming aspect of the course such as game developing using certain language, alternative approaches which jump out of the traditional reorganization, that is, teaching programming language means teaching student to design and develop software using that language, and backtrack to the original critical question in computer science itself, that how to mimic/magnify human intelligence/behavior through machine, will probably attract more attentions from students and expand their research interests.

An environment that compiles students' code written in specified programming language or predefined framework to automaton and visualizing the automaton in user-welcome interface can be applied for the purpose described above. By providing simple programming interfaces that users can define logics for autonomous entity and present the behaviors corresponding to the defined logics under a game context, students can get immediate satisfaction as playing general games. On the other hand, while designing and implementing automatons, the students won't miss any traditional components including data-flow, data structure, simple algorithm, problem abstracting and basic knowledge of programming language itself in the course. Such system aims at increasing users' programming skill, help them understand data structure/algorithm, extend their interests in exploring the basic problem sets of computer sciences.

Existing work of systems in this category focused

Manuscript received July ,26, 2011; revised August 13, 2011; revised August 16, 2011

M. Wang is currently a Phd student in Department of Computer Science, Georgia State University, Suit 1407, 34 Peachtree Rd, Atlanta, GA ,30303, emali: csloader@gmail.com

X. Hu is an associate professor in Department of Computer Science, Georgia State University, 34 Peachtree Rd, Atlanta, GA ,30303 email: xhu@cs.gsu.edu

heavily on theoretical and formal environment for A.I design and programming language. In RoboCode [10], developers created a tank-war game environment with a java compiler to compile users' codes as intelligences for tanks in a 2-D virtual battlefield. The rule is simple, the tanks with their logics defined by user fight with each other and the finally survived one (or team) wins the game. The project is used world-widely as an industrial level java language learning tool in professional trainings for new employee by many IT companies. Alternatively, it also owns large number of fans among academic researchers who are interested in A.I field. They even hold global tournament for competing and evaluating new A.I theories using this platform. On the other hand, BZRobots [11] is a project which has the similar concept with RoboCode but has a 3D graphic simulator. In the field of robot soccer, the organizers of Robot Soccer Cup also provide a simulation environment for the competitions that can run without a hardware robot [12]. This software simulated the environment and equipments as a real robot soccer matches while the design of internal logic that how to move, turn or kick for each robot is left for user. This project is academic and rigorous in a sense of researching rather than education. With formal programming language constraints and complex simulator, almost all of these systems are definitely designed for professionals with fairly enough programming experiences and advanced knowledge of A.I algorithm. And the mental model of these systems does not fit the mental model of novice students who are more likely to be attracted by interesting and competitive tasks.

On the other hand, pure education oriented systems exist as well. For example, the project Institute for Personal Robots In Education (IPRE)[8] has shown great advances in attracting students interests by providing them an environment that they can define the robot's behaviors and interactions with environment using java programming in CS1 courses. Comparing to the robot soccer match or virtual battlefield, a single robot is much easier for student to try ad play with. Such real-time system is intuitive and attractive enough for new students to deliver their interests, and is very preferable for the universities that are able to afford the hardware financially.

Inspired from the previous work mentioned above, in this paper, we present SoccerCode, an agent-based, competitive soccer game system designed for novice students who majored or have intentions in studying computer science/engineering. The reason of introducing the form of soccer game in our project is that as a game, it is interactive and interesting. As mentioned, for an educational system used in CS1 courses, attracting learner's interests and motivating their interactions with each other is critical. A strict and complex system requires convincible skills could be interactive but may fail to stimulate the learner's interests. On the other hand, a stand-along modeling intensive approach could be interesting but may lack the components of games such as competitions and interactions because the learner won't recognize the strength of their work or gain enough satisfactions. Based on agent-based modeling theory, SoccerCode provides an interactive but straightforward game environment that users can define the intelligence for virtual soccer players. The team who makes more goals on opponent wins at the end of the game. The system provides flexible mechanism and various JAVA based control APIs to simplify the procedure of designing behaviors of autonomous agents and initializes a virtual robot soccer test bed for learners to examine wellness of their design by conducting match with each other. Furthermore, aiming at providing an integrated educational tool, our system has includes an evaluating subsystem that enable batch simulations to evaluate the strength of a student design. The basic idea is to hold matches between teams designed by students and one that designed by the teacher, the performance of student's team will be evaluated through certain criteria such as number of goals and loss. By providing such features, this system can be simple enough but keeps possibility for deeper exploration, where the progress of students can be evaluated by the teachers easily as well. The rest of this paper will be organized as follows. In section 2, the theoretical reference we used for this project is presented. We demonstrate the abstract specification of the system in section 3. In section 4 we show detailed functionality and use cases while section 5 provides an example of using the system for educational purpose. We conclude the current work and propose potential trends of future development of our project in section 6.

II. SYSTEM OVERVIEW

An agent, defined in the area of computer science, is an autonomous entity with its own decision-making mechanism and knowledge-base depends on which it performs accordingly behaviors, interacts with outer world and retrieves feedbacks. In our system, the context is a 2D ground with two virtual teams on it. Each soccer agent is an agent with capability to perceive environment, memorize states and perform particular actions; it has properties such as position in the 2D ground, speed of moving, direction it heads to and various user-defined properties to memorize the states of the environment. It also provides a programming interface for defining the internal logic of each agent on the ground. After the logics are defined, users can watch the match by running a simulation in the graphic simulator to see performance of their coded logics. There are three types of matches provided by the system, 1v1, 2v2, or 3v3, with respectively 1, 2, 3 soccer agents on each side. The lesser agents on the ground, the easier logic of each of them is required to perform reasonably because multiple agents require higher complexity of cooperation and strategy. Whenever users finish designing their teams, they can save their team to local disk as an xml file. By loading saved teams to the ground, a match can be hold between two users to compare the strength of their A.I design. We also provide a pre-designed standard team for each of the 3 game categories; this team is used to evaluate users' designs. The evaluation is done by loading users' teams and putting them onto the virtual ground against the standard teams. The results (goals and goals conceded the users' teams get) are the bases of the final evaluation of the team design. This process of evaluations won't be shown in the 2d graphic simulator as normal matches, but users can replay them and investigate the problems by loading the log file of the evaluations.

The provided system allows user to define properties such as name, initial position of the agents in a user welcome interface.Robot soccer players are driven by a set of behaviors defined by users. Each behavior is composed by control code that using APIs provided by the system. These APIs are basically a set of functions which users can control the agent by calling them in user's code. They include various functions such as measuring the distance between two objects (soccer player/goal/ball), perceiving the position of objects, memorization and performing actions such as moving and kicking the ball. The APIs restrict the power of users' codes and are standard of actions that both teams are able to perform to make the game fair and impartial.

For the knowledge-base (which means the knowledge of each agent about the environment, history and other agents) part, we use a layered structure to organize the agents' knowledge-base. The knowledge-base for each agent have been categorized into three layers: The first layer is *global*, which records the system information and is accessible by all soccer agents of both two teams on the ground and cannot be modified by user's code. The second layer is *team*, which is only accessible for members of each team. *Private* is the third layer whose accessibility is limited within a single individual agent. Using the latter two layers, users are able to program their soccer agent to memorize the information, system status or experience so as to develop clever and tactical soccer agents. This architecture provides a hierarchical organization of knowledge-base for each agent; furthermore, it follows the concepts from domain of programming language to help learners get a better understanding about the importance of access control.

a. Programming Interface

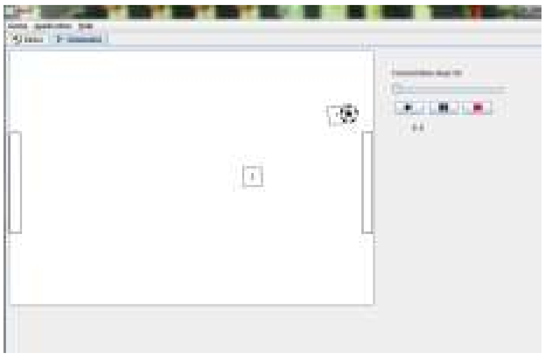
b. Game Play Interface

Figure 2.1 Interfaces

Figure 2.1 presents the basic interface of the system. The programming interface provides text editor that user can write codes for the behaviors of each robot soccer. The editor provides highlighting keywords and simple syntax check. After the behaviors of the robot are defined, users can watch the match in 2d graphic simulator shown in Figure 1b. The example shows a 1 vs 1 match.

III. SYSTEM ARCHITECTURE

To demonstrate the system specification, it is important to introduce the concept of behavior network. For decision making of each agent, one major concept SoccerCode adopted is behavior network architecture. Behavior Network [1][3] architecture embodies the concept of behavior control, one of the most important control paradigms for adaptive autonomous agents. Within the behavior network of each agent, the behaviors are treated as independent computational models. Typically, a behavior has two modules of codes, one is *excitation* and the other is *action*. The excitation module is a module that returns threshold value representing that how a behavior can be excited by external or internal state changes. The action module is the actual action an agent will perform after one behavior has been excited. In a single agent, if the coefficients between behaviors are assigned to non-zero value, these behaviors form a behavior network which defines how they affect each other to carry out the final behavior to perform. We use mutual inhibition/excitation mechanism for the behavior network connections in our work. It defines coefficients upon the network connections. For each simulation step, the agent selects its behavior to perform as follows,

a. Calculate basic activation for behavior j using Activation(j)=Excitation(j)
b. Calculate affected activation for behavior j using Activation(j)=Activation(j)+SUM(i , Activation(i)*Coefficient(i,j))
c. Select the behavior with highest activation

That is: only the behavior which has highest calculated activation can be performed. This mechanism is straightforward, flexible and skips complex logic flow control as traditional approach. Also, by defining single behavior instead of creating a behavior network, users can still turn back to traditional ways to write procedure-based code for their soccer agents.

A. Formulization

We first specify the formulization of system model. We denote uppercase initials as variable of a set while lowercase initials as variable of atomic property.

The system of can be described as follows

S={Pr, A, O} (1)

Where Pr is a set of all the properties that available from system, and holds that

Pr={P_api, P_env, P_rule} (2)

Formula (2) is interpreted as:

P_api={getPosition(), setDirection(), getDirection(), kick(), move(),……..}

Where P_api os the set of APIs that available for user to set up their agents' model

P_env={v_friction, v_collision, v_mspeed..}

Where P_env is the set of physical attributes of the environment such as Friction, Collision and Speed limit.

P_rule={r_power, r_accuracy, r_anticipation..}

Where P_rule is the set of coefficient that determines the randomness and uncertainty of such property the simulation agents: Power, Accuracy, Anticipation..

Another element, A, in formula (1) is the set of the user agent that need to be defined by the users. It contains two subsets: A_team1 and A_team2, each of which represents the collection of agents in the same team on the square:

A={A_team1, A_team2} (3)

A_team1=A_team2={Agent, team_knowlege_base} (4)

Where team_knowledge_base is the user specified information only accessible to the agents in the same team

Agent={Agent1,Agent2,Agent3}

Agent1=Agent2=Agent3=

{position, speed, direction, individual_knowledge_base, BehaviorModel} (5)

Where an agent's individual_knowledge_base is its knowledge about the environment that only accessible to itself

BehviorModel={BehaviorNetwork, procedure} (6)

procedure is a singleton behavior that the agent will always perform which provides a different way from behavior network for user to define the agents' internal logic.

BehaviorNetwork={Behaviors, connection, coefficent} (7)

Where connection and coefficient defines how behavors interact with each other.

Behaviors={Behavior1,Behavior2…..Behaviorn}

Behaviorn={action, excitation}

Where action and excitation defines how the behavior will be activated and what action the agent will take when it actives this behavior.

B. *Data Flow.*

SoccerCode is a discrete time based system. In each time step, the simulator thread executes the user defined code of each agent and calculates its position, speed and direction. It then updates the knowledge-base(User defined variables) if specified in the defined code. The graphic engine will then refresh the interface subsystem using the values updated. A history record vector is also provided to store a log of the match procedure and export necessary values to some particular APIs such as getDistanceBetweenTwoEntities(), etc.

For each agent, the execution state transition differs between behavior network mode and single procedure mode. In behavior network mode, the agent behaves as follows: It executes user defined code in the excitation subsystem of each behavior to determine the initial excitation; it then calculates the final exicitation value using behavior network and the coefficiences within it; finally it selects the behavior which has the highest excitation; and executes user defined action code in the selected behavior model. The state transition of behavior network agent can be shown as figure 3.1:

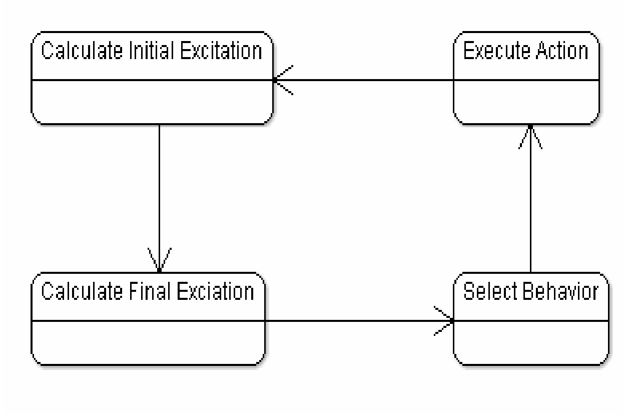

Figure 3.1

In single procedure mode, the agent simply executes the action code in each step, which is similar to the traditional way of A.I design. Shown as 3.2:

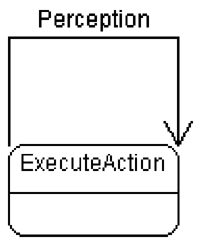

Figure 3.2

C. *System APIs.*

One educational purpose of SoccerCode is to establish the interests and motivation for learners to explore their potentials on programming or computer science related topics. To achieve this purpose, we export various system APIs with functionalities of controlling agent motion, measuring the environment perception or creating dynamic knowledge base to users so they can easily design the logic. The APIs are categorized into three area: Control APIs, Measurement APIs and Memory APIs.

a. Control API

APIs in this category can be used to control the motion and actions of the agent.

Typical instances include:

robotMove(direction, speed);

//move the agent who called this method towards some direction with some speed

kick(direction, speed);

//kick the ball if the ball is near to the agent who called this method to some direction with some speed

b. Measurement API

These set of APIs are rules and locators that are likely to be used as the perception. Without these APIs, agent can hardly observe the environment. Useful examples are:

getMyId();

//return the identifying number of the agent who called this method

getDirection();

//return the current of the agent who called this method

getDistanceBetween(int Object1, int Object2);

//return the distance between one object and another (either goal, player or ball).

```
getSpeed(int Object);
//return the speed of an object.
```

c. Configuration API

The variable APIs can be used as to create agents memory system which stores the information of the environments, including all accessible states of other objects. This set of APIs is useful if user intend to challenge a more complex A.I design such as self-adaptive agents or reinforcement-learning agents:

```
createVariable(name, type);
//create a variable with a name and type
setVariable(name, value);
  //set the value of a created variable
```

As an example of using these APIs, consider the situation that we want the soccer agent kick the ball towards the goal-line of the opposite team when the distance between the ball and target goal-line is shorter than 100. The code of this behavior should like follows:

```
If (getDistanceToOppsGoal()<100.0){
      If (ballOnFoot()){
       kick(getDirectionToOppsGoal(), MAX_SPEED);
}
}
```

Using these simple APIs, users can quickly get involved in the design procedure and begin their lessons on control flow and basic programming skill by themselves.

D. Environment

Another subsystem of SoccerCode is the environment configuration subsystem. In this subsystem, the physical features can be configured according to users' requirement. Currently, the configurable features include friction, which will affect the motion of the ball object; collision, which describes when two agents' body are engaged, what effect will be applied to their motions. On the other hand, to provide variety and uncertainty of the match procedure, we introduce noise parameters which are automatically applied to corresponding APIs. For instance, the *kick power* determines the uncertainty of power in the action "kick", which means the actual speed of the ball after it is being kicked is a random value based on the power user sets in the action "kick". The *kick accuracy* has the similar effects but determine the direction that user set to the "kick" action. This subsystem is mainly responsible for making the match more realistic and flexible.

E. Evaluation

As an educational tool, we integrated an evaluation subsystem to evaluate the progress of students in learning through the game. We defined a standard team that has basic behavior for each of its member agent and a non-trivial strategy; we then run matches in batch-executing mode using the standard team and users' team. The score will be evaluated according to the average performance of user's team. All the evaluating procedure is saved so that the matches can be replayed in the 2D ground if necessary

IV. Functionalities

SoccerCode aims at providing a friendly and interactive interface for easy play. In this section, we briefly describe the functionalities of the system.

A. Team Configuration

In SoccerCode, there are three match modes currently available: 1vs1, 2vs2, 3vs3. The 1vs1 mode, which assigns one single soccer agent to each team, is for beginners while the later modes are involved with multi-agent and are more complex. The match is held on a 2D ground. The 2D ground is divided into two areas; a drag-drop ground panel is provided for setting the position of agent players for each team. By selecting an agent player, the behavior model of that agent will be listed as well as the basic information of that agent .Initially, all the agent is represented by the name "Team"+team_side+player_id, and has an empty behavior model. To create new internal logic of the agent, simply select an agent and create a new behavior using the behavior network editor. To define a new variable of the agent, you can either use createVariable() API in your behavior bnmodel or just create it manually in the team configuration panel.

B. Behavior Network Editing

In the pop-up panel of behavior network editor, one can create, delete, modify behavior/behavior network model of a single agent. A semantic and symbol checking mechanism is also provided in the coding area which gives users a report about what semantic or symbolic error they have made when their code is not able to get approved and compiled by the system.

C. Environment Setting

To change the value of friction, collision or the random coefficients, one can access the configuration panel , this configuration is also available during simulation procedure and will immediate affects the simulation.

D. File Management and Model Parsing

One requirement of the system is that when user finished their designing procedure, they can save their model as a xml file and load their saved models to the square so as to compete with others' or consistently improve their model in the future. For this purpose, a model parser is provided to convert from a team model to an xml file or from an xml file to a team model. All internal logic, knowledge base and player information are included in the file, i.e, a trained neuron network or reinforcement learning data map in agent model is portable and can be migrated to another platform without information loss.

E. Simulation Control.

SoccerCode uses a 2-D graphic simulator as an engine to simulate the match and present it in 640*480 animation frames. During the simulation procedure, one can stop/pause the match or restart/resume it at any time stamp. The simulation stops and is then reset if there's a goal in either side on the square. The goal and scorer will be recorded and users can then adjust their agent model, reconfigure their position and restart the match.

Next section describes how to use these functions to develop a simple 1 player team with the match mode 1v1.

V. Demonstrative Examples

In this section, we give a demonstrative example of developing a simple soccer agent in a 1 on 1 game to show the functionalities of the system. In SoccerCode, to define a team it always involved with several steps: First, choose a mode from 1v1, 2v2 or 3v3. Note that each team model is not compatible among three modes; second, for all agents in as single team, either on the right or the left side of the square, define their behavior control logic with the API system provides. This can be done by either loading saved team files from disk to each side or create new teams and programming their logic in the interface provided; at last, users can store their team as an xml file for competition purpose or future improvements.

In our demonstrative examples, we design an agent without behavior network, that is, simple procedure based agent for a 1vs1 match. We set up the game mode to 1 vs 1 mode, that is, each team of the match will have only one agent.

We then create the behavior "Attack Forward Run" in the behavior editor and use the APIs we supported to control the action of the agent according to its observation.

Here we define the logic as follows; when the agent is not close enough to the goal, it simply moves towards goal and at the same time avoid players of other side and try to maintain the ball around itself. If the agent is close enough to the goal, then it stop maintaining the ball and simply kick it to the goal with max speed.

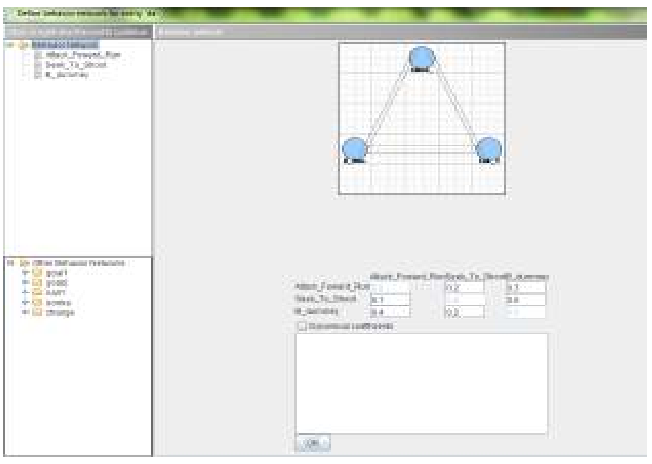

Figure 5.1 Design the relationship between behaviors

By decomposition the single procedure we mentioned above, a behavior network version of the example is shown as follows; the conditions and its action are defined as single behavior and connected with network. This will enforce the behavior selection mechanism for decision making. This mechanism is actually more flexible in a sense that it provides more dynamic interactions among components of the behavior controls.

Finishing designing procedure, one can adjust the position within half of the square owned by the team or configure the environment variables such as friction or randomizing coefficients and switch to simulation panel to match.

VI. Conclusion

In this paper, we present SoccerCode, an agent based soccer simulation game system that can help motivate interests and intentions to participate of learners in computer related area. A hierarchy of the system architecture is defined and each system component is described in different levels, either abstractly or practically. We have incorporated this system as one of the tools for teaching Java programming in the course CSC2310: Principles of Programming in Spring 2011 at Georgia State University. Students learned how to set up a 1v1 robot soccer game and to program basic movements of robots using the Java language taught in the class. They showed strong interests in the tool and provided positive feedbacks. We plan to incorporate this tool in future CSC 2310 courses and formally assess how effective the tool is in helping students' learning. Furthermore, to improve the usability and interactivity of the motivation of this project we described, we aim at distributing it to the web based platforms for further improvement.

References

[1] A. Marino, L. E. Parker, G. Antonelli, and F. Caccavale, "*Behavioral Control for Multi-Robot Perimeter Patrol: A Finite State Automata Approach*", Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 2009

[2] D. Ranum,, B, Miller., J. Zelle,., M. Guzdial,. Successful Approaches to Teaching Introductory Computer Science Courses with Python, Special Session, SIGCSE'06, March 1-5, 2006, Houston, Texas, USA.

[3] F. Qiu, X. Hu, *BehaviorSim: A Learning Environment for Behavior-based Agent*, Proc. The 10th International Conference on the SIMULATION OF ADAPTIVE BEHAVIOR (SAB'08), 2008

[4] G. M. Schneider, "*The introductory programming course in computer science: ten principles*", ACM SIGCSE Bulletin - The papers of the SIGCSE/CSA technical symposium on computer science education Homepage Vol 10 Issue 1, February 1978

[5] K. Becker "*Teaching with games: the minesweeper and asteroids experience*," J. Comput. Small Coll, 17(2), 23-33, 2001

[6] S. Leutenegger, J. Edgington, "*A Games First Approach to Teaching Introductory Programming*" Proceedings of the 38th SIGCSE technical symposium on Computer science education, vol 39, issue 1.

[7] S. Zweben *Undergraduate CS Degree Production Rises; Doctoral Production Steady,* Computing Research News, 2011, vol 23 No 3

[8] T. Balch; L Summet and others, *Designing Personal Robots for Education: Hardware, Software, and Curriculum,* Pervasive Computing 2008, Vol 7, issue 2 , pp. 5–9.

[9] T. Lorenzen., W. Heilman,, "*Cs1 and cs2: write computer games in java!* " SIGCSE Bull., 2002 vol 34 issue 4 , pp. 99-100,

[10] The RoboCode Project, available : http://robocode.sourceforge.net

[11] The BZRobot Project, available: http://my.bzflag.org/w/Main_Page

[12] The RoboSoccerCup Simulator, available: http://sourceforge.net/apps/mediawiki/sserver/index.php?title=Main_Page